Facultat d'Informàtica de Barcelona (FIB)

UNIVERSITAT POLITÈCNICA DE CATALUNYA (UPC)
BarcelonaTech

Treball de Fi de Grau:

# EMULACIÓ DE L'ALGORISME D'ENCAMINAMENT DE LA XARXA MESH SENSE FILS GUIFI.SANTS

MEMÒRIA

Autor: Joan Pol Alejandre Obiols
Data de la defensa: 26/06/2018

---

Director: Llorenç Cerdà (Tecnologies de la informació)

---

Grau en Enginyeria Informàtica
Especialitat en Tecnologies de la Informació

**Resumen**

Vivim en un mon en constant evolució. Amb el creixement exponencial de les tecnologies i les infraestructures, cada cop es més difícil testejar nous protocols sobre sistemes en producció. D'aquí sorgeix la nostra necessitat de simular escenaris com els reals, amb el propòsit d'avaluar nous protocols o tecnologies sense cap perill, i d'aquesta manera, poder estar evolucionant constantment.

L'objectiu d'aquest projecte es simular la xarxa mesh sense fils Guifi.sants a l'emulador de xarxes CORE, i conèixer fins a quin punt aquest emulador s'adapta a les nostres necessitats de simulació.

# Índex

# Índex de figures

# 1. Formulació del problema

Guifi.net és una xarxa comunitària que des de fa uns anys s'ha anat fent popular sobretot a Catalunya i Països Valencians, encara que s'està començant a expandir per tot el món. Cada cop són més els usuaris que opten per aquest tipus de xarxa, en la qual es comparteixen els recursos entre els usuaris que la formen per tal de crear un accés a Internet diferent del que havíem conegut fins ara.

Guifi.Sants és l'àrea de la xarxa comunitària Guifi.net que delimita el barri de Sants.

Amb el constant creixement de les tecnologies i la globalització, cada cop és més important no deixar de banda tant el rendiment com la seguretat a les nostres xarxes. És per això necessari estar constantment millorant i desenvolupant noves tecnologies que ens permetin evolucionar en termes d'eficiència i rendiment a les nostres xarxes.

L'objectiu d'aquest TFG és emular la xarxa Guifi.sants amb un emulador de xarxes de computadors, per tal de poder avaluar el rendiment, simular el seu algorisme d'encaminament, i provar si cap nous protocols i topologies de xarxa. Per exemple, recentment s'ha desenvolupat una nova versió de l'algorisme d'encaminament que fa servir la xarxa sense fils de Guifi.Sants, per tant poder avaluar abans aquest algorisme en un entorn virtual resultaria molt útil.

# 2. Contexte

## 2.1. Guifi.net

Guifi.net[1] és una xarxa de telecomunicacions **oberta**, **lliure** i **neutral** que s'estén amb el tram que incorpora cada participant al connectar-s'hi.

- **Oberta** perquè tothom hi és convidat a participar. Sense discriminacions ni secrets, tothom pot conèixer com funciona i millorar-la.
- **Lliure** perquè no hi ha un propietari que imposa les seves condicions. Permet comunicar-se amb els altres membres de la xarxa i escollir com accedir a internet.
- **Neutral** perquè l'acord d'interconnexió és entre iguals i afecta només a la xarxa i no pas als continguts que hi circulen.

## 2.2. Xarxes comunitàries

Les possibilitats que ofereix la tecnologia de comunicacions de dades sense fils ha facilitat que arreu s'hagin construït múltiples iniciatives de xarxes sense fils. Ja sigui per us particular, comunitari o de tota mena, com també per a connectar amb el veïnat i així poder posar en comú recursos i serveis. Com que aquestes xarxes no estan sota el control de una empresa amb ànim de lucre, només coneixem el límit de les possibilitats tecnològiques, la imaginació i la capacitat de cadascú.

L'objectiu de la **fundació guifi.net** és doncs posar en comú aquesta infraestructura i proporcionar els mecanismes d'organització perquè funcioni i es gestioni. No és en cap cas una empresa ni privada ni governamental orientada a proporcionar un servei públic o de pagament, sinó simplement la coincidència en un interès comú del grup de persones que el forma i al qual tothom hi és convidat a afegir-s'hi. Enfocat d'aquesta manera, els beneficis que proporciona són notables: Avui en dia la tecnologia crea excedents en els recursos que no sempre podem aprofitar de forma particular, aleshores, si els posem en comú, donem accés altres persones que a la vegada estan disposats a compartir amb nosaltres.

Aquesta lògica no només és simple, sinó a més té un component molt important en el món d'avui en dia: Crear un espai on s'extreu un benefici particular però precisament per fer-ho d'una manera col·lectiva. És una manera de fer servir les tecnologies en benefici de les persones i no pas a l'inrevés.

Les comunitats wifi compleixen també una altra funció important, i és la de posar a l'abast de tothom tecnologies tan aviat com algú estigui disposat a fer-ho. No pas només a conveniència d'estratègies comercials de grans empreses, que no sempre actuen segons les condicions de mercat i possibilitats tecnològiques, sigui per posició dominant o en funció de la regulació existent. I al preu que costa, no pas al que ens diguin.

guifi.net també és un espai de recerca i desenvolupament per a l'adaptació d'aquestes tecnologies a les finalitats que li són pròpies en una xarxa oberta.[2]

## 2.3. Guifi.net i Internet

A Guifi.net hom forma part d'una xarxa de comunicacions electròniques oberta que no restringeix l'accés a ningú. Com a xarxa oberta, gui-

fi·net forma part de la xarxa de xarxes Internet. Les connexions entre guifi·net i Internet són múltiples i poden adoptar formes variades.

La xarxa, com a xarxa oberta, lliure i neutral té vocació inequívoca de ser una xarxa més de la gran xarxa de xarxes que és Internet. Per aquest motiu, guifi.net procura formar part activa dels organismes que formen Internet i, com a operador, procurar acords i mecanismes d'interconnexió eficients amb tots els altres operadors. Tanmateix, actualment, a Internet, els altres operadors poden ser més restrictius i no tan oberts pel que fa a la interconnexió amb les seves xarxes, per exemple fent una explotació comercial de la interconnexió o del trànsit.

Com que en aquests casos no és possible aplicar el principi de reciprocitat, el trànsit cap a la internet d'aquests operadors es considera fora de l'àmbit del Comuns de la Xarxa Oberta Lliure i Neutral (XOLN)[3], i com a tal, respecte de la XOLN, n'esdevé un contingut. Per tant, aquesta mena d'interconnexió s'ofereix als participants com qualsevol altre servei dins de la xarxa, i típicament n'hi pot haver de dos tipus:

- El que proporcionen els participants a partir de les connexions que puguin tenir amb altres operadors com a portes cap a Internet.
- El que s'obté a través de la interconnexió amb xarxes ("peering") d'altres operadors.

Qui proporciona aquests serveis determinarà si ho fa "tal qual", o amb algun tipus de compromís de servei.

## 2.4. Guifi.Sants

Guifi.sants[4] es un projecte que té com a objectiu portar i posar a disposició de les veïnes del barri de Sants l'accés a la xarxa guifi·net.

A la tardor del 2009 va néixer el projecte guifi·sants, com iniciativa popular per estendre al barri de Sants la xarxa alternativa, lliure, oberta i neutral guifi·net.

El dissabte 13 de març de 2010, l'equip guifi·sants va muntar el primer supernode del barri i el 10 d'abril del 2010 el barri de Sants es va connectar a la xarxa lliure guifi·net. Es va enllaçar el primer SuperNode amb l'Hospitalet, passant d'aquesta manera a formar part de l'estructura troncal de la xarxa. Aquesta acció es va portar a terme gràcies a les primeres voluntàries de Sants de la mà de l'assemblea del barri de Sants, la gent de L'Hospitalet Wireless i el grup guifi de

Figura 1: Ampliació de la xarxa Guifi.Sants

Riereta.

A principis de l'any 2011 es va apostar per una nova configuració de la xarxa: el model de xarxa en malla, on cada node client és també un node repetidor. Aquest nou model va permetre una expansió més ràpida i efectiva de la xarxa a Sants. Com més veïnes hi ha connectades, més fàcil és connectar-ne de noves.

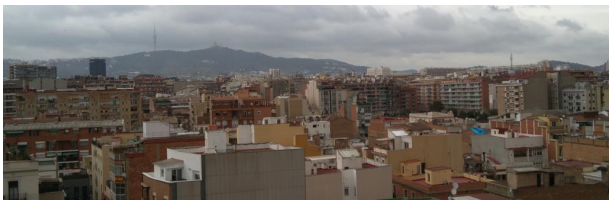


Figura 2: Sants (xarxa Guifi.Sants)

A principis del 2017 hi ha més de 70 nodes operatius a la xarxa de Sants. i tothom està convidat a afegir-se i ampliar-la.

En aquest projecte es tractarà d'emular la xarxa de Guifi.Sants a l'emulador de xarxes Common Open Research Emulator, per a després poder portar a terme tot tipus de simulacions amb l'objectiu de millorar la xarxa.

## 2.5. Actors implicats

Les persones que es beneficiaran del resultat d'aquest projecte són principalment els administradors i els usuaris de la xarxa Guifi.Sants. Facilitant una eina que emuli la seva xarxa es podran portar a terme moltes simulacions que mostrin com es comportaria aquesta en diferents escenaris, per a després escollir la millor opció.

# 3. Estat de l'art

L'únic que s'ha trobat en relació a projectes anteriors semblants a aquest és una prova en el que s'emulava una xarxa mesh ad-hoc[6] (s'explicarà més endavant), el qual aconsegueix fer una xarxa simple i establir comunicació entre els nodes, i, encara i fer servir BMX6 com a algorisme d'encaminament, el propòsit i el sistema encastat és diferent del que volem aplicar en aquest projecte.

Donat que no s'han trobat molts projectes anteriors semblants a aquest, es procedirà a explicar l'estat actual de les tecnologies que es faran servir en aquest projecte.

## 3.1. Xarxes mesh sense fils

Una xarxa sense fils mesh és una xarxa de comunicacions feta a partir de nodes sense fils organitzats en topologia de malla.

El terme malla es refereix a una connexió completa o quasi completa entre els nodes de la xarxa. Els nodes estan interconnectats entre si per a tal d'establir moltes possibilitats i camins possibles, això fa una xarxa consistent i evita l'aïllament de molts nodes en cas de caiguda d'algun d'aquests.

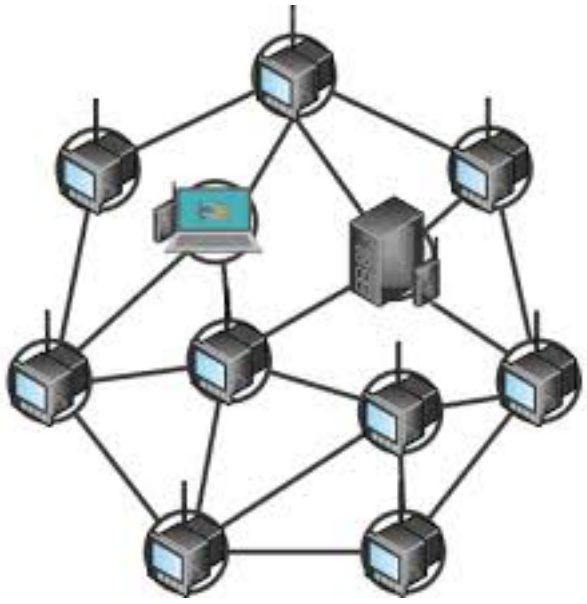


Figura 3: Exemple de xarxa tipus mesh

En el cas de les xarxes sense fils mesh, els dispositius són bàsicament dispositius sense fils com antenes, repetidors wifis, etc. que poden accedir al medi sense fils, mentre que els routers encaminen el tràfic des de i cap a les altres xarxes externes, com pot ser Internet.

L'accés a aquest tipus de xarxa depèn del radi dels nodes que formin part d'aquesta, treballant en harmonia els uns amb els altres per a crear un radi acceptable. Una xarxa en malla és fiable i ofereix redundància. Quan un node cau o no pot operar, la resta de nodes es podran comunicar entre ells igualment directament a través d'algun altre node intermediari.

Les xarxes mesh sense fils es poden formar soles o restaurar-se de la mateixa manera, ja que estan constantment actualitzant la informació de la xarxa, per a poder actuar de pressa i actualitzar els camins en cas de caiguda d'un dels nodes.

## 3.2. Algorisme d'encaminament BMX6

BMX6[7] (BatMan-eXperimental version 6), el sucesor de "BatMan-eXperimental", és un protocol d'encaminament mesh per a xarxes comunitàries, i el que fa servir la xarxa de Guifi.Sants. Agafant característiques i havent après d'altres protocols com OSLR, Babel, i Batman, aquest protocol relativament jove incorpora fonamentalment nous conceptes d'arquitectura i mecanismes adreçats als nous reptes de les xarxes wifi comunitàries.

El disseny i desenvolupament d'aquesta versió és el resultat de l'objectiu de fer front i d'incrementar l'espai d'adreces donat per IPv6, activar configuracions de node-individual mentre que es clarifica el maneig d'anunciaments de nodes conflictius (per exemple, duplicats d'IP), i també permet una disseminació d'estats eficient entre la distinció d'estats "localï "globalçom també entre "dinàmicï .estàtic".

BMX6 es fa servir activament a grans CNs (Community Networks) com guifi.net, Graciasensefils[8], Freifunk[9], i Lugro-mesh.[10]

BMX6 és un protocol d'encaminament per taules per a xarxes mesh sense cables. Com qualsevol protocol d'encaminament per taules, el seu objectiu és compondre un camí des de l'origen fins al destí, decidint cada node el qual serà el següent pas. És un protocol de vector de distància, ja que la informació que cada node gestiona és una llista de tuples d'identificador de nodes i el cost fins a arribar al determinat node, havent-hi escollit un determinat link. .

La novetat de BMX6 és el mecanisme de disseminació que fa servir per a propagar aquesta informació. El protocol de disseminació està inspirat en xarxes socials humanes que són escalables, ja la gent tendeix a aprendre més sobre els seus veïns, i abstrau i filtra més informació dels altres que no coneix.
El coneixement de la topologia a un node està optimitzat per al node en contret i per als seus veïns, fent servir identificadors locals compactats per a un diàleg en estat comprimit.

Durant la fase transitòria, els veïns intercanvien dades sobre el seu entorn: descripció de nodes, links, etc. I proveeixen informació dels seus identificadors individuals (IIDs), que identifiquen els nodes d'una manera compacta. Amb aquesta informació, cada node construeix una taula de diccionaris per veí que tradueix les seves IIDs al hash global únic i no-ambigu de la descripció del node. Durant l'estat estable, cada node té informació local amb el format de diccionari IID-to-hash, i la informació global de l'estat com a diccionari hash-to-description.

Durant aquesta fase, el protocol simplement intercanvia petits paquets per a mantenir l'historial de mètriques i monitorar els canvis de la xarxa. Gràcies a la informació d'estat que s'ha aconseguit durant la fase transitòria, aquests camps poden ser substituïts per IIDs més petits (16bits), el que fa que aconseguim missatges més comprimits.[11]

### 3.3. Protocol d'encaminament OSPF

Open Shortest Path First (OSPF) és un altre protocol d'encaminament molt conegut, i bastant semblant a BMX6, el qual busca el camí més curt entre tots els nodes i estableix la taula d'encaminament amb els millors camins. [5]

### 3.4. Quick Mesh Project

Els nodes de la xarxa de Guifi.Sants fan servir la distribució qMp[12] (quick mesh project), un sistema encastat en els nodes basat en Linux, en el qual està integrat l'algorisme d'encaminament BMX6, entre d'altres més funcions necessàries.

Quick Mesh Project (qMp) és un sistema per desplegar xarxes Mesh/-MANET fàcilment emprant tecnologia Wi-Fi.

Figura 4: Exemple de un node qMp

- És un firmware encastat per a dispositius de xarxes, basat en el sistema operatiu OpenWrt/LEDEL.
- Proporciona una manera fàcil per a configurar xarxes MESH, sense importar si aquestes són cablejades o Wireless, o una fusió de les dues.
- És una manera fàcil i rentable d'estendre Internet als usuaris finals.

qMp ha estat dissenyat per ser utilitzat en tota mena d'escenaris com xarxes comunitàries obertes, xarxes corporatives, grans esdeveniments públics, desplegaments de xarxa ràpids, etc.

El firmware qMp, basat en on OpenWrt[13], una distribució Linux per a sistemes encastats completament lliure i que es pot modificar completament, funciona en diversos dispositius WiFi encastats.

Tot el codi font de qMp és públic, lliure i obert, i s'hi pot accedir des de la pàgina de desenvolupament.

## 3.5. Common Open Research Emulator (CORE)

El "Common Open Research Emulator" CORE es una eina per a emular xarxes en una o més màquines. Es pot connectar aquest emulador a xarxes reals. CORE consisteix en una GUI per a dibuixar topologies de màquines virtuals lleugeres, i mòduls Python per a emular scripts de xarxa.

CORE ha sigut desenvolupat per un grup de recerca de tecnologia de xarxes que es part del "Boeing Research and Technology division". CORE es un projecte de la U.S. Naval Research Laboratory[14]

Les seves característiques més notables son:

- Laboratori de xarxes "in a box"
- Eficient i escalable
- Interface "easy-to-use"
- Configuració i control centralitzats

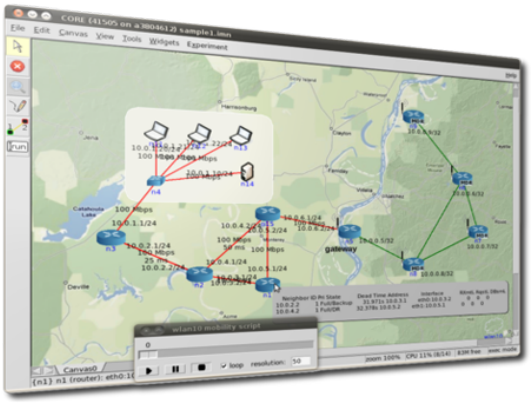


Figura 5: Core Project

- Executa aplicacions i protocols sense modificar-los
- Connexió real-time per a xarxes reals
- Hardware-in-the-loop
- Software **lliure i obert**

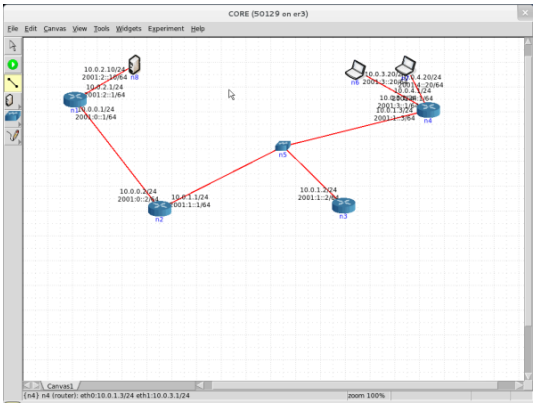


Figura 6: Captura de pantalla de interfície de Core

# 4. Abast

Per tal de simplificar el projecte, el dividirem en dos objectius. El primer consisteix en muntar l'escenari que volem a l'emulador de xarxes, i el segon simular l'algorisme de la xarxa mesh.

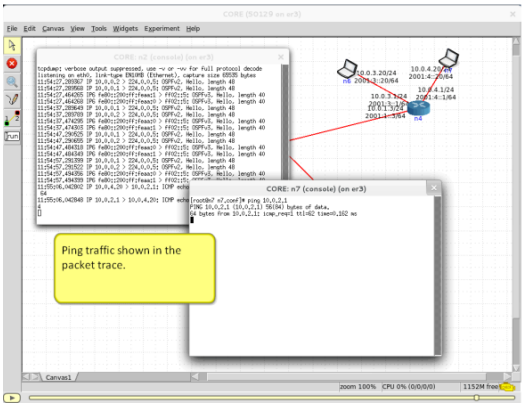



Figura 7: Captura de pantalla de la interfície de CORE

## 4.1. Primer objectiu

El nostre objectiu principal és entendre l'emulador de xarxes CORE en profunditat, i analitzar si realment ens pot proporcionar les eines de simulació necessàries per a representar el tipus d'escenari que volem. Si això no fos possible, s'estudiarien altres emuladors de xarxes per a veure si s'adapten a les nostres necessitats.

## 4.2. Segon objectiu

El segon objectiu és simular l'algorisme d'encaminament de la xarxa mesh. Idealment amb BMX6, el qual està dins de qMp, igual que a la xarxa real, però si aquest al final no es pot fer servir, es farà servir un de semblant, en el nostre cas OSPF6, ja que ve integrat a CORE.

Per a poder tenir un escenari més realista, els nodes de CORE haurien de córrer la distribució qMp que es fa server en la xarxa Guifi.sants. Integrar un node qMp en CORE pot ser una tasca complicada, doncs la distribució qMp és molt diferent de la distribució que porten els nodes integrats en CORE. Com no sabem quant de temps pot portar aconseguir aquest objectiu, aquest s'intentarà al principi, però si no s'assoleix en un temps determinat, el deixarem com a *future work*. Aleshores, passarem a simular amb l'algorisme OSPF6.

Si s'aconseguís córrer qMp a l'emulador, i es comprovés que el funcionament d'aquest és com el de la xarxa real o almenys se li assimila, es podria provar la nova versió de l'algorisme BMX7, i avaluar les mesures de seguretat que implementa. Seria interessant, ja que assolir aquest objectiu voldria dir que l'emulador compleix molts dels requisits

que desitgem, i per tant el projecte encara tindria més sentit.

# 5. Metodologia i rigor

## 5.1. Mètode

El mètode serà simple, es crearà un entorn virtual amb CORE i es començarà primer a intentar una xarxa simple, després anirà evolucionant si tot va bé fins a formar una xarxa com la de Guifi.Sants.

S'aniran executant els blocs de tasques i cada cop que es tinguin resultats nous o configuracions noves, es pararà a avaluar el resultat i veure si es va per bon camí.

Les fases del projecte estan definides a la secció de tasques, en la qual es veuran els grans blocs que s'han de anar realitzant per a progressar.

## 5.2. Eines

Donat que el projecte serà individual, l'única eina que es farà servir serà Dropbox per a compartir els arxius, i poder seguir treballant en qualsevol lloc si no es disposa de l'ordinador principal i s'està treballant amb el portàtil. Per a la comunicació amb el director es farà servir e-mail.

## 5.3. Validació

Per a validar el funcionament i els resultats es programarà una cita amb el director del projecte, per a comprovar que efectivament s'estan aconseguint els resultats tal i com els ha proposat el director, i veure si estem plasmant la xarxa igual que la de Guifi.Sants.

Per al resultat dels estudis, primer es veurà quin tipus de prova vol realitzar el director, i després es presentaran els resultats i es podran treure les conclusions adients a partir d'aquests.

Encara no esta establert en quin format es presentaran els resultats dels estudis perquè primer s'ha de veure si es poden fer, però probablement serà generant PDFs.

# 6. Planificació temporal

A continuació es planificarà el temps segon les tasques que definirem del projecte, així com també els recursos i la valoració d'alternatives.

## 6.1. Durada del projecte

La planificació d'aquest projecte té una duració estimada de 4 mesos, començant al febrer i acabant al juny. L'estimació de les hores totals es d'unes 425 hores, aproximadament. S'iniciarà primer la fase de planificació, i després s'aniran executant les tasques fins a arribar a la conclusió final i presentació al torn de juny, si no hi ha cap contratemps. En aquest cas el projecte es podria allargar fins al següent quadrimestre, però no es l'objectiu.

## 6.2. Tasques

Dividirem les tasques en petits blocs que ens ajudaran a assolir millor totes les etapes del projecte. Aquestes aniran desde l'exploració del software d'emulació de xarxes virtuals, proves i recreació del escenari de Gufi.Sants, fins a l'actualització del algorisme d'encaminament i documentació final del projecte, si els punts anteriors s'han arribat a assolir.

PT1: Emulador de xarxes virtuals CORE: Entendre el funcionament de CORE, i de quina manera podem començar a treballar amb aquest.

- T1.1 Identificar l'envirorment de CORE
- T1.2 Muntar l'entorn necessari per a CORE
- T1.3 Descarregar el source i instal·lar
- T1.4 Tractar de muntar una primera xarxa bàsica
- T1.5 Entendre el seu funcionament

PT2: Xarxa mesh: En aquest escenari es tractarà de muntar una petita xarxa que tingui topologia mesh i analitzar el seu comportament. En el cas de que el resultat sigui satisfactori, es procedirà a emular la xarxa Guifi.Sants a la següent tasca.

- T2.1 Entendre la topologia de xarxes mesh
- T2.2 Construir una petita xarxa mesh
- T2.3 Comprovar que el seu funcionament es efectivament el de una xarxa mesh

PT3: Quick Mesh Project: entendre el seu funcionament, i veure com podem encastar el sistema dins d'una maquina virtual de CORE

- T3.1 Entendre el funcionament de qMp
- T3.2 Descarregar el source i tractar d'encastar-lo a una maquina virtual CORE
- T3.3 Montar una xarxa de prova amb les maquines qMp

PT4: Emular la xarxa Guifi.sants: En aquest punt es tractarà d'emular de forma el més aproximada possible la xarxa mesh de Guifi.Sants. Això implica poder especificar els enllaços que hi ha entre els nodes, i les seves característiques.

- T4.1 Entendre estructura Guifi.Sants
- T4.2 Recursos que ens serveixen a CORE
- T4.3 Muntar la xarxa
- T4.4 Avaluar el resultat

PT5: Simulació de l'algorisme d'encaminament de la xarxa mesh. Idealment qMp, l'algorisme que es fa servir a Guifi.Sants, sinó es farà servir OSPF6, ja que es el més semblant i ja està integrat a CORE.

- T5.1 Definir l'algorisme d'encaminament a l'emulador
- T5.2 Testejar funcionalitat
- T5.3 Definir proves
- T5.4 Simular i avaluar els resultats

PT6: Documentació i presentació

- T6.1 Escriure la documentació final
- T6.2 Preparar la presentació

## 6.3. Dependències

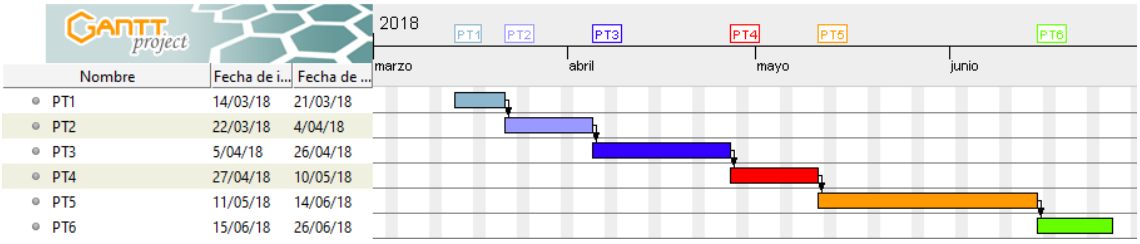
El projecte el realitzarà una sola persona, per tant l'execució de les tasques serà seqüencial. Totes les tasques tenen com a dependència la tasca anterior, per tant no es realitzarà un diagrama de Pert. Cal dir que una tasca que si que es realitza durant el projecte es la de documentació i preparar la presentació, ja que ho fem a GEP, però com al final també s'ha de redactar bastant es comptabilitzara al final.

### 6.4. Temps estimat de les tasques

| Tasca | Hores |
|---|---|
| PT1: Emulador de xarxes virtuals CORE | **10** |
| T1.1 Identificar l'envirorment de CORE | 1 |
| T1.2 Muntar l'entorn necessari per a CORE | 2 |
| T1.3 Descarregar el source i instalar-lo | 2 |
| T1.4 Tractar de muntar una primera xarxa basica | 2 |
| T1.5 Entendre el seu funcionament | 3 |
| PT2: Xarxa mesh | **15** |
| T2.1 Entendre la topologia de xarxes mesh | 5 |
| T2.2 Construir una petita xarxa mesh | 5 |
| T2.3 Comprovar que el seu funcionament es el de una xarxa mesh | 5 |
| PT3: Quick Mesh Project | **60** |
| T3.1 Entendre el funcionament de qMp | 10 |
| T3.2 Descarregar el source i tractar d'encastar-lo a una maquina virtual CORE | 40 |
| T3.3 Muntar una xarxa de prova amb les maquines qMp | 10 |
| PT4: Emular la xarxa Guifi.sants | **150** |
| T4.1 Entendre estructura Guifi.Sants | 10 |
| T4.2 Recursos que ens serveixen a CORE | 20 |
| T4.3 Muntar la xarxa | 100 |
| T4.4 Avaluar el resultat | 20 |
| PT5: Simulació de l'algorisme d'encaminament | **130** |
| T5.1 Definir l'algorisme d'encaminament a l'emulador | 40 |
| T5.2 Testejar funcionalitat | 50 |
| T5.3 Definir proves | 5 |
| T5.4 Simular i avaluar els resultats | 35 |
| PT6: Documentació i presentació | **60** |
| T6.1 Escriure la documentació final | 50 |
| T6.2 Preparar la presentació | 10 |
| | |
| Total | **425** |

Taula 1: Desglossament de la estimació dels blocs de tasques.

## 6.5. Diagrama de Gantt




## 6.6. Recursos personals

L'únic recurs personal del que es disposa en aquest projecte es un treballador, el mateix autor del projecte. Aquest treballador realitzarà totes les funcions per a poder portar a terme el projecte.

El temps que es destinarà semanalment al projecte serà aproximadament 30 hores setmanals, depenent de les dificultats que puguin aparèixer.

## 6.7. Recursos materials

1. *Ordinador portàtil MSI GP72M 15"pulsades.* Eina de hardware per al desenvolupament del projecte. Amb aquest es farà servir el software de virtualització i es documentarà tot el projecte, així com també totes les comunicacions necessàries i consultes a bases de coneixements o professors.

2. *Overleaf.* Eina de software online per a documentar el projecte. Es un editor de latex que mostra el output en directe per pantalla, el que facilita molt la redacció del document.

3. *Dropbox.* Eina de compartició d'arxius cloud en la que es compartiran tant els documents com els entorns que es vagin desenvolupant.

4. *Roundcube Webmail.* Eina de correu de la FIB que es farà servir per mantenir la comunicació amb el director del projecte.

5. *CORE.* Software de emulació de xarxes virtuals en el qual s'emulara la xarxa Guifi.sants

6. *Connexió a internet.* Eina hardware essencial per a la elaboració del projecte i per a poder fer servir totes les eines esmentades anteriorment.

## 6.8. Valoració d'alternatives i pla d'acció

Tal i com està descrit al diagrama de GANTT, el projecte està planejat per a acabar el dia 18 de juny aproximadament, ja que s'escollirà el torn de juny per a fer la defensa d'aquest, i s'ha d'entregar la memòria una setmana abans de fer la defensa.

Durant el transcurs poden sortir alguns imprevistos que poden alentir el procès. Com be s'ha explicat en l'abast del projecte, dividirem aquest en dos objectius, el primer es aconseguir muntar la xarxa de Guifi.Sants a CORE, i el segon seria emular l'algorisme d'encaminament BMX6, i si no es pot, fer servir un com ara bé OSPF.

El risc més notable és que no puguem encastar qMp a les màquines de CORE, un risc que assumim amb un 30 % de probabilitat. Donat la poca documentació i la dificultat del procés de implementació.

Per a aquest primer tipus d'imprevist estimarem unes 30-40 hores de més aproximadament.

El segon risc que considerem es que realment CORE sigui més difícil de configurar del que ens pensàvem, a l'hora de representar la xarxa de Guifi.Sants. Donat que la documentació no és molt extensa i no hi han molts estudis al respecte, això pot arribar a alentir la investigació. No obstant, el risc d'aquest imprevist es preveu de un 5 %.

Per a aquest segon tipus d'imprevist estimarem unes 40 hores de més aproximadament.

# 7. Sostenibilitat i Pressupost

## 7.1. Autoavaluació sobre la sostenibilitat

Durant el transcurs de la carrera a la facultat, hem anat tractant temes d'importància envers la sostenibilitat, com ara bé l'obsolescència programada, el consum energètic eficient, residus directes i indirectes generats per les TIC, materials utilitzats i procedència d'aquests, o l'accessibilitat als medis TIC.

Crec que és molt important conèixer bé aquests factors i ser conscient d'ells, ja que ens trobem en un punt de la nostra evolució molt delicat, un punt en el qual hem de començar a canviar molts aspectes començant per l'actuació local i des de tots i cada un dels projectes als quals ens enfrontem, doncs crec que és molt encertat aquell dit que diu "Pensa localment, actua globalment". Al cap i a la fi planeta només hi ha un, i ja està més que demostrat que hem arribat a punts els quals ja no podem mitigar el mal causat.

Gràcies als seminaris i diferents competències transversals a través del curs, hem sigut capaços de ser conscients d'aquests factors, i, per tant, poder observar amb indicadors els diferents aspectes mediambientals i socials als nostres projectes, i aplicar les solucions més adients per a reduir al màxim les seves conseqüències.

Un aspecte que també s'ha tocat bastant (sobretot a alguna assignatura opcional com ara CPD's) és la gestió econòmica d'un projecte. Costos variables, fixes, amortitzacions, etc. són aspectes que et fan ser conscient de la importància de la bona gestió econòmica als projectes. Ja que, al final, els diners són un factor molt important.

Quant al tema social, potser és un dels temes que considero menys hem emfatitzat durant la carrera (almenys a la nostra especialitat), per tant almenys per a mi no més tan fàcil identificar aquests aspectes. Per sort el meu projecte és sobre una xarxa oberta, el qual m'ha fet veure que alguns projectes estan molt orientats a l'aspecte social, i així adonar-me'n que a vegades no tot és el benefici propi i la producció amb fins econòmics.

En resum crec que s'ha tractat molt bé tots els temes de sostenibilitat tant ambiental, com social i econòmica durant el curs. Ara, com tot, fa falta començar a posar-ho en pràctica.

## 7.2. Dimensió econòmica: Pressupost

A continuació es farà una previsió dels costos aproximats del projecte. Es tindran en compte els recursos humans, recursos de hardware i també els possibles imprevistos, per a tal de tenir una estimació global la més aproximada possible a la realitat.

### 7.2.1. Recursos Humans

A l'hora de valorar els costos dels recursos humans s'ha tingut en compte que només serà un treballador el que dugui a terme tot el treball, per tant l'únic cost seran les seves hores.

En un principi s'ha pensat de dividir per rangs el tipus de treball, i assignar un preu per hora diferent a cadascun segons la importància del seu treball. Però donat que totes les tasques son similars en termes de dificultat i realment son molt semblants, no considerem que faci falta diferenciar-les per rangs.

Per tant una única persona serà l'encarregada de portar a terme totes les tasques explicades anteriorment a la secció de tasques. Per tant les despeses en recursos humans quedarien, aproximadament, d'aquesta manera:

| Tasca | Hores | Cost |
|---|---|---|
| PT1: Emulador de xarxes virtuals CORE | **10** | **200** |
| T1.1 Identificar l'entorn de CORE | 1 | 20 |
| T1.2 Muntar l'entorn necessari per a CORE | 2 | 40 |
| T1.3 Descarregar el source i instal·lar-lo | 2 | 40 |
| T1.4 Tractar de muntar una primera xarxa bàsica | 2 | 40 |
| T1.5 Entendre el seu funcionament | 3 | 60 |
| PT3: Xarxa mesh | **15** | **300** |
| T2.1 Entendre la topologia de xarxes mesh | 5 | 100 |
| T2.2 Construir una petita xarxa mesh | 5 | 100 |
| T2.3 Comprovar que el seu funcionament es el de una xarxa mesh | 5 | 100 |
| PT3: Quick Mesh Project | **60** | **1200** |
| T3.1 Entendre el funcionament de qMp | 10 | 200 |
| T3.2 Descarregar el source i encastar-lo a CORE | 40 | 800 |
| T3.3 Muntar una xarxa de prova amb les maquines qMp | 10 | 200 |
| PT4: Emular la xarxa Guifi.sants | **150** | **3000** |
| T4.1 Entendre estructura Guifi.Sants | 10 | 200 |
| T4.2 Recursos que ens serveixen a CORE | 20 | 400 |
| T4.3 Muntar la xarxa | 100 | 2000 |
| T4.3 Avaluar el resultat | 20 | 400 |
| PT5: Simulació de l'algorisme d'encaminament | **130** | **2600** |
| T5.1 Definir l'algorisme d'encaminament a l'emulador | 40 | 800 |
| T5.2 Testejar funcionalitat | 50 | 1000 |
| T5.3 Definir proves | 5 | 100 |
| T5.4 Simular i avaluar els resultats | 35 | 700 |
| PT6: Documentació i presentació | **60** | **1200** |
| T6.1 Escriure la documentació final | 50 | 1000 |
| T6.2 Preparar la presentació | 10 | 200 |
| | | |
| Total | **425** | **8500€** |

Taula 2: Estimació de costos de recursos humans.

Tenint en compte que la mitja de totes les activitats desenvolupades són de 20 l'hora, tenim un total de **8500€** totals estimats per als recursos humans.

### 7.2.2. Recursos materials

En aquest apartat s'inclouran els costos dels recursos materials de hardware i software del projecte. Es consideraran les eines que es fan servir habitualment per treballar, que son bàsicament es un ordinador, i el software adient per a portar a terme totes les proves.

| Producte | Preu | Unitats | Vida útil | Hores | Amortització |
|---|---|---|---|---|---|
| MSI GP72M | 1242 | 1 | 4 anys | 425 | **103,5€** |

Taula 3: Estimació dels recursos materials.

### 7.2.3. Software i llicències

En aquest apartat no es tindrà cap cost addicional, donat que el sistema operatiu utilitzat serà Windows 10 i ja ve de sèrie a la màquina esmentada anteriorment, així com també Microsoft Office pel Word i Excel. Si s'ha de fer servir un altre sistema operatiu per a fer la virtualització serà Ubuntu 16.04, i aquest és gratuït. Tot el software addicional que es farà servir, com Dropbox per compartir arxius, Overleaf per a generar els documents PDF, VirtualBOx per a virtualitzar en el cas que es necessiti, etc. serà gratuït. Per tant no s'estima cap cost addicional per a software i llicències.

### 7.2.4. Altres costos

Tindrem en compte dos aspectes més a l'hora de calcular els costos, un cop calculats els recursos humans, software i hardware:

- **Costos indirectes:** Com que tota l'activitat serà portada a terme al domicili del treballador, es generaran costos addicionals com l'habitatge (es farà servir l'habitació com a oficina de treball), i també l'energia, aigua, i internet. Això té un cost total de 350€al mes. Durant 5 mesos, fan un total de 1750€. Però com només es farà servir aproximadament 8 hores al dia els dies laborables, això suposa 160 hores al mes, que són un 23,8 % de les hores totals del mes, per tant el cost indirecte serien **416€**.
- **Imprevistos:** Haurem de tenir en compte també els possibles imprevistos que s'han comentat anteriorment a l'apartat de *riscs*. En el pitjor dels casos el que pot passar es que s'hagi de buscar un altre software de virtualització o que es perdi més temps del compte fins a aconseguir encastar el sistema en les màquines, o intentant recompilar els kernels un cop s'hagi implementat el nou algorisme BMX7. Comptarem amb 10 dies més de treball el que fan 80 hores, i un total de **1600€.**
- **Contingència:** Per a cobrir qualsevol cost en la variació del projecte, tant com per imprevistos com per qualsevol risc, assumirem un nivell de contingència del 15 % sobre la suma dels costos totals directes i indirectes. Per tant els costs de contingència són **1592€**.

### 7.2.5. Resum de costos

| RRHH | Hardw. | Soft. | Indirectes | Conting. | Imprevistos | Total |
|---|---|---|---|---|---|---|
| 8.500€ | 103,5€ | 0€ | 416€ | 1592€ | 1.600€ | **12.211,5€** |

Taula 4: Estimació dels recursos materials.

## 7.3. Control de gestió

El control de les despeses en aquest projecte es bastant simple, bàsicament perque el cost principal es el temps e l'únic treballador. Per a controlar aproximadament el temps i no desviar-se molt de la previsió, es realitzaran periòdicament reunions amb el director del projecte, ja bé quan s'hagi assolit una nova fase del projecte, o quan es porti bastant de temps estancat en un mateix punt. Amb aquestes reunions es podrà veure si realment s'està fent servir el temps de manera adient, i si es podrà arribar en el temps estimat a realitzar totes les tasques del projecte.

En el pitjor dels casos el director del projecte podrà arribar a veure qualsevol pèrdua de temps en un problema que no es podrà resoldre o que potser portarà massa temps la solució, i aportarà les eines necessàries o les pautes per a que es torni a anar en la direcció correcte.

## 7.4. Sostenibilitat i compromís social

## 7.5. Econòmic

- **Reflexió sobre el cost estimat per al projecte**
  Crec que els costos estimats son bastant aproximats. Per al que fa els recursos humans, si la definició de tasques i del temps empleat per a aquestes es bastant aproximat, simplement haurem de contar les hores per a portar a terme les tasques i no hi ha més complicació en aquest aspecte. Pel que fa al hardware també es bastant fàcil, ja que només es farà servir l'ordinador del treballador del projecte. Els costos indirectes i els imprevistos crec que també son bastant adients, donat també per la seva senzillesa en l'àmbit dels indirectes i els imprevistos una miqueta per lògica, i per tenir un marge considerable de error. Amb una miqueta de sort en el resultat final es pot arribar a reduir els costos.

- **Com es resolen actualment els aspectes de costos del problema que vols abordar?**
  En aquest aspecte no es soluciona cap tema de costos, ja que el projecte en si tracta d'evolucionar un aspecte tècnic de la xarxa comunitària, i també cal dir que els costos estan minimitzats per aquest

motiu, per tant no crec que es pugui resoldre cap cost perquè es pràcticament inexistent.

- **En què millorarà econòmicament (costos...) la teva solució respecte a les existents?**
  Al tenir la capacitat de simular diferents escenaris amb diferents protocols, podem millorar la xarxa en molts aspectes, com ara bé la seguretat. Si millorem la seguretat d'una xarxa podem reduir els costos econòmics, com per exemple reduint els robatoris online.

## 7.6. Ambiental

- **Has estimat l'impacte ambiental que tindrà la realització del projecte?**
  L'únic impacte ambiental que pot tenir el projecte es el consum energètic que pot tenir l'equip al realitzar-lo, a part d'aquest no hi ha cap mes. Per tant l'impacte ambiental es mínim sino innexistent (ja que igualment l'ordinador es faria servir per altres projectes i estaria encès).

- **T'has plantejat minimitzar-ne l'impacte, per exemple, reutilitzant recursos?**
  No, ja que l'impacte es mínim. Es mes, si potser s'intentés re-utilitzar un ordinador vell, el consum podria arribar a augmentar donat a que al ser un model anterior el consum podria ser menys eficient i per tant augmentaríem el consum energètic.

- **Com es resol actualment el problema que vols abordar (estat de l'art)?, i. En què millorarà ambientalment la teva solució respecte a les existents?**
  A nivell ambiental no es proposa cap problema i per tant s'en resol cap. No es pot millorar cap aspecte ambiental en la xarxa actual de Guifi.Sants amb aquest projecte, ja que tracta únicament l'aspecte software. Com a molt, es podria arribar a dir que, gràcies a les simulacions i a la implementació de nous protocols o algoritmes, els dispositius treballen amb menys càrrega donat que son més eficients, i, per tant, reduirien el seu consum energètic.

## 7.7. Social

- **Què creus que t'aportarà a nivell personal la realització d'aquest projecte?**
  A nivell personal podré ampliar els meus coneixements actuals de xarxes, així com conèixer més a fons el funcionament d'una de les xarxes comunitàries més grans d'Europa. Abans de fer la carrera vaig fer un cicle superior de xarxes, ja que m'agradaven, i per les xarxes també vaig accedir a aquesta especialitat. Per tant a nivell personal m'agrada molt acabar la carrera amb un projecte exclusivament de xarxes. Un aspecte que també m'està agradant i que em costava molt al principi és el concepte de preparar un projecte i desglossar-lo per tasques abans de començar, entendre tots els passos i analitzar els riscos, com també analitzar els aspectes econòmics, socials i ambientals.

- **Com es resol actualment el problema que vols abordar (estat de l'art)? i. En què millorarà socialment (qualitat de vida) la teva solució, respecte a les existents?**
  En termes de millora, si el nou algorisme funciona millor, farà possible que els usuaris de la xarxa comunitària tinguin una xarxa més eficient, i per tant millorarà el funcionament de la infraestructura, donant millor connexió als usuaris i en general un millor servei de la xarxa.

- **Existeix una necessitat real del projecte?**
  Potser en aquest aspecte, el projecte està destinat a millorar una solució ja existent, per tant es podria dir que, si no es realitzés el projecte, la gent podria seguir fent servir la xarxa com fins ara. Per tant no hi hauria una necessitat real, a no ser que en un futur es trobessin amb problemes d'eficiència donat a un augment físic de la xarxa, i es necessités un algorisme més eficient.

# 8. Coneixent Common Open Research Emulator

El primer grup de tasques tracta de conèixer l'emulador CORE, i preparar l'entorn per a poder treballar amb aquest.
El que fa a CORE un emulador de xarxes interessant es el fet que emuli els nodes de la xarxa i, a més a més, simuli el medi físic, aproximant-se bastant a la realitat. S'ha de tenir en compte que en alguns casos com a les xarxes sense fils, es molt difícil emular l'entorn real, ja que pot ser que hi hagin edificis o elements de l'entorn afectin al medi i que a l'emulador no es poden arribar a tenir en compte.

## 8.1. Funcionament

CORE emula la xarxa internament amb mòduls de Python, i, encara i poder córrer terminal interactiva de cada node quan la emulació està en marxa, el core dels nodes corre realment al host, això vol dir que les llibreries dels serveis s'agafaran del node host. Per exemple, si volem executar Quagga, un dels serveis per defecte als nodes de CORE, haurem de instal·lar Quagga a la màquina host per a que aquests puguin trobar les llibreries.

El servei s'administra a través del CORE daemon en backend, per a emular les sessions. El daemon es controla des de la interfície gràfica, denominada CORE GUI. La comunicació entre el daemon i la GUI es fa a través d'una API personalitzada basada en sockets, anomenada CORE API.

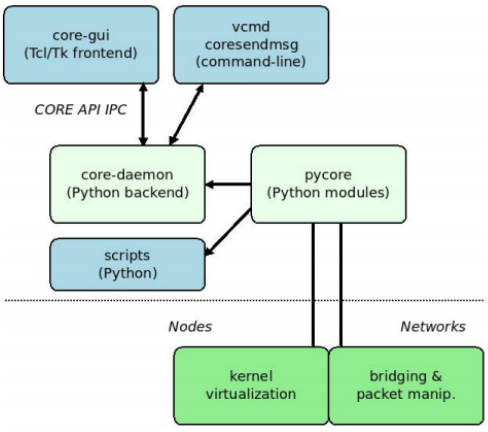



Figura 8: Arquitectura de CORE

## 8.2. Linux containers

La tècnica principal de virtualització de CORE es basa en Linux Containers, també anomenats LXC o netns. Les distribucions de Linux més recents com Fedora o Ubuntu tenen kernels preparats per a aquests contenidors, d'aquesta manera el kernel no ha de ser recompilat ni modificat.

L'espai de noms per als contenidors es crea fent servir la trucada a sistema *clone()*. Cada espai de noms té el seu propi entorn de procés i una pila de xarxa privada. Els espais de noms de xarxa tenen el mateix sistema de fitxers en CORE.

CORE combina aquests espais de noms amb *Linux Ethernet bridging* per a formar les xarxes. Les característiques de les connexions entre els nodes s'apliquen fent servir protocols d'encaminament *Linux Netem*.

## 8.3. Modes d'operació

La interfície gràfica de CORE té dos modes principals de funcionament, *Edit* i *Execute*. Si correm la comanda *core-gui* sense cap opció addicional a una terminal, iniciarem el *Edit* mode. Els nodes es dibuixen en un llenç blanc fent servir la barra d'eines a l'esquerra, i es configuren fent clic dret o doble clic a sobre del node. La interfície gràfica no necessita ser executada com a *root*.

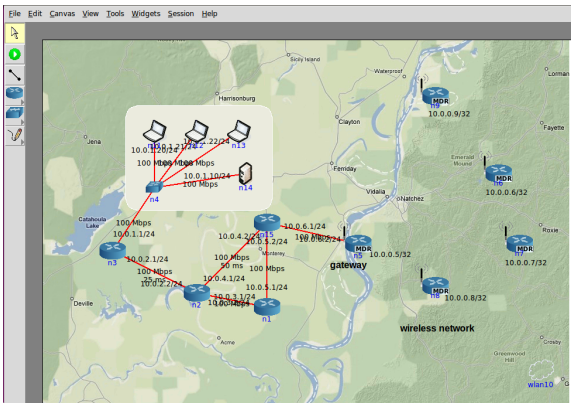



Figura 9: Mode d'edició de CORE

Es genera la topologia desitjada, si es pren el botó *Start* s'executa el mode d'execució. En aquest mode, l'usuari pot interactuar amb amb les màquines que s'estan emulant, fent doble clic o botó dret al node desitjat.

Un cop dintre del mode d'execució la barra d'eines es convertirà en mode execució, oferint eines d'execució. Si prenem el botó vermell *Stop*, passarem un altre cop al mode *Edit*.

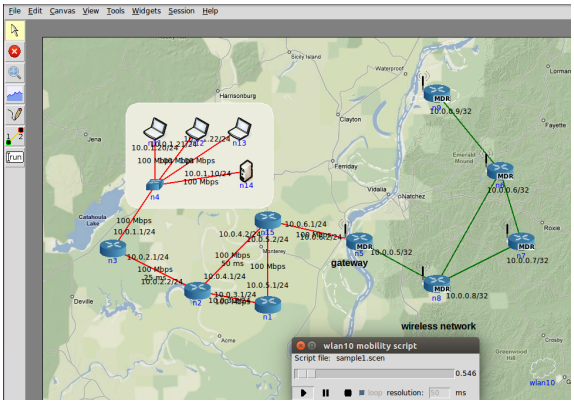



Figura 10: Mode d'execució de CORE

## 8.4. Fitxers de configuració

En tot moment podrem obrir un entorn prèviament guardat en un fitxer amb l'extensió ".imn", la qual conté tota la informació necessària de la xarxa per a poder ser editada i executada a dins de CORE. Podem veure que es un llenguatge de configuració propi de CORE.

```
node n2 {
    type router
    model router
    network-config {
  hostname n2
  !
  interface eth0
   ip address 10.0.1.1/24
   ipv6 address 2001:1::1/64
  !
    }
    canvas c1
    iconcoords {274.0 157.0}
    labelcoords {274.0 189.0}

}
```

Figura 11: Exemple de la declaració d'un node

A més a més dins del fitxer .imn estan definides totes les xarxes WAN i demès dispositius a part dels nodes, com també totes les característiques addicionals de la xarxa.

En aquest fitxer es on s'haurà de plasmar la topologia de la xarxa de Guifi.sants, traduint la informació de la que es es disposa dels nodes com ara bé la posició x i y respecte al mapa, la seva adreça IP, etc.

### 8.5. Tipus de node

Tenim tres tipus de node que podem fer servir a CORE:

- **Netns**: Aquesta es la maquina per defecte, fa servir Linux containers. Aquest tipus de màquina es molt lleugera, i necessita una quantitat mínima de virtualització per a emular una xarxa, apart de que per a fer la virtualització no fa falta ni modificar ni re-compilar el kernel, ja que es farà servir el de la màquina host.
- **Físic**: Els nodes físics s'utilitzen per a representar màquines reals basades en Linux que formaran part de la xarxa. Aquest tipus de node es fa servir per exemple per a incorporar servidors i endpoints a la xarxa. No farem servir aquest tipus de node i per això no entrarem en més profunditat.
- **Xen**: Les màquines xen son bastant interessants per a aquest projecte, ja que permeten córrer qualsevol tipus de instància de sistema operatiu que tingui Xen al core. Per desgràcia Xen a CORE està en fase experimental, amés només accepta imatges ISO, i la documentació es limitada com també la funcionalitat.

## 9. Quick Mesh Project a CORE

En l'apartat de tasques 2 tracta de entendre el sistema encastat Quick Mesh Project (qMp), i intentar-lo encastar en una màquina virtual CORE.

En un principi no havia de ser molt complicat, però a mesura que s'ha anat investigant i provant, i, donat que la documentació per a poder implementar aquest tipus de sistema a CORE es molt escassa, s'han acabat tenint problemes, el que ha derivat a una conclusió qual canvia una part del projecte, i la qual s'explica en els següents apartats.

## 9.1. Implementació amb Xen

Xen es un monitor de màquines virtuals que executa instàncies de sistemes operatius amb totes les seves característiques a un host, fent servir molts pocs recursos. Existeixen distribucions Linux que ja venen amb Xen de sèrie, com ara bé RedHat, CentOS o Debian.

Com que qMp esta basat en OpenWrt, i aquest es una distribució amb kernel Debian, Xen sembla una bona opció per a córrer màquines virtuals a CORE amb instàncies de qMp.

El primer problema que es troba al intentar implementar aquesta idea es que CORE només accepta format .iso per als nodes Xen. Es descarrega una imatge de qMp (versió Clearance, la versió de la majoria dels nodes de la mesh de Sants), i es prova de convertir a .iso, per a implementar-la a CORE.

No s'aconsegueix que la imatge s'executi un cop pasa de .img a .iso, el que ens fa pensar que potser exactament aquesta versió de qMp no porta integrat Xen al kernel. Es comenta amb el director i aquest proposa de provar OpenWrt, ja que es la distribució pare.

Es fan diverses proves amb imatges de OpenWrt que teòricament han de tenir suport per a Xen, però al ser aquestes també format .img, i al no ser capaços de generar una .iso a partir del source, no sembla tasca fàcil. Després de provar varies opcions, i, donat que la documentació per a implementar Xen a CORE es molt escassa i també és una funció experimental de l'emulador, es decideix provar altres vies.

## 9.2. Implementació amb contenidors Docker

Una altre opció que sembla interessant es fer córrer contenidors Docker als nodes, i executar una instància de OpenWrt dins. Amés Docker sembla tenir una imatge de OpenWrt a la seva wiki ja preparada per a descarregar i executar, el que faria tot més fàcil un cop Docker estigués corrent al node, ja que només hauríem d'importar la imatge i iniciar-la.

El tipus de node a CORE serà Linux Container (netns). Seguint els pasos d'un estudi que s'ha trobat amb Core i Docker (només s'ha trobat això de documentació per a aquest tipus d'implementació), es crea un script python dins de la carpeta de configuració dels serveis de CORE que referencia al servei de Docker que prèviament haurem compilat a la màquina host, ja que realment s'executarà a través del Docker de la màquina host.

També s'haurà d'afegir una línia al script d'inici del daemon de CORE per a poder afegir Docker com a servei.

```
1  ""myservices
2
3  Custom services that you define can be put in this directory.  Everything
4  listed in __all__ is automatically loaded when you add this directory to the
5  custom_services_dir = '/full/path/to/here' core.conf file option.
6  """
7  __all__ = ["sample", "docker"]
```

Figura 12: Modificació de __init__.py a per afegir Docker als nodes

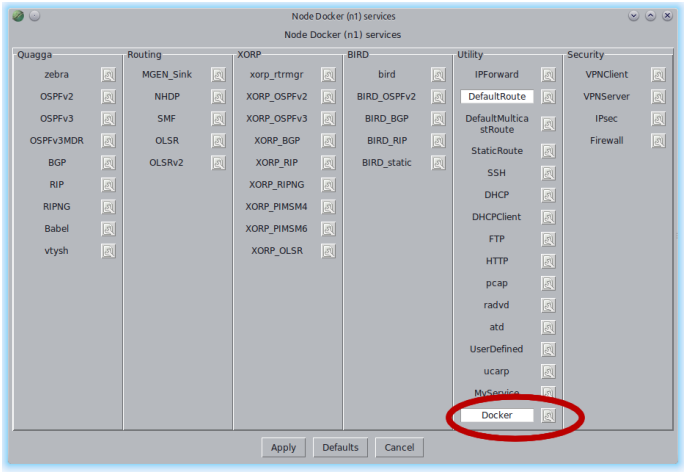



Figura 13: Servei Docker disponible als serveis del node

Tot i haver preparat l'escenari com sembla ser la manera correcte, al intentar còrrer una simulació de prova amb nodes netns sembla ser que el servei Docker no inicia correctament dins del node. El daemon Docker està corrent a la màquina host, pero per alguna raó no aconsegueix vincular-se amb el de la màquina host.

Finalment no s'aconsegueix fer córrer Docker als nodes, donat que només es disposa d'aquest estudi i la documentació es inexistent, potser portaria massa temps aconseguir el resultat i tampoc sabem després si podríem córrer OpenWrt.

Aquestes dues opcions eren les mes assequibles per a executar qMp a CORE, i, al trobar-se amb la dificultat tècnica, a continuació s'avaluarà si potser es podria córrer a altres simuladors en comptes de CORE.

# 10. Explorant altres simuladors de xarxes

Després d'haver intentat encastar qMp als dispositius de CORE sense èxit, es realitza una reunió amb el director del projecte, el qual proposa provar altres simuladors, per veure si potser era mes fàcil d'implementar.

## 10.1. Mesh Linux Containers

Mesh Linux Containers (MLC), es un packet del creador del protocol d'encaminament BMX6 que fa servir Linux containers en xarxa per a simular el seu comportament. El que fa interessant a aquest paquet es que fa servir uns recursos mínims, el qual fa assimilar el seu comportament més a la realitat.

Es fan algunes proves per veure si es pot córrer qMp a un contenidor, i, tot i semblar possible, veiem que realment MLC requereix de molta instal·lació manual, amés el paquet fa anys que no s'actualitza i la documentació es quasi be inexistent. Això, junt amb el fet de que aquest simulador tampoc disposa d'interfície gràfica, fet bastant important per nosaltres, ens fa descartar-lo, ja que la inversió en temps d'aprenentatge seria massa elevada i els resultats potser no serien tan visuals. Per tant, descartem MLC com a una opció donava la seva dificultat i la manca d'interfície gràfica.

## 10.2. Mininet

Mininet es una eina bastant millor documentada, molt feta servir a la comunitat de *software defined networking* (SDN). Pot crear xarxes de computadors virtuals connectats a un commutador SDN basat en OpenFlow.

OpenFlow es una tecnologia de commutació que permet a un servidor de software determinar el camí de reenviament de paquets que hauria de seguir una xarxa de commutadors. Des d'aquest software es controlen tots els nodes de la xarxa, i aquest també simula el comportament del medi.

En aquest cas també sembla possible que es pugui fer córrer qMp, però un altre cop ens trobem sense interfície gràfica. També s'hauria d'aprendre a fer servir OpenFlow, el qual està ben documentat, però el temps que portaria aquest aprenentatge sumat a la manca de GUI fan que també descartem aquesta opció.

### 10.3. GNS3

GNS3 es un emulador de xarxes molt versàtil, ben documentat i amb moltíssimes opcions ja pre-configurades en quant als nodes. Sembla una eina molt interessant sobretot perquè ja disposa de nodes amb implementació OpenWrt, i tant la documentació com la interfície gràfica és molt completa, donant un ampli ventall d'opcions i tipus de nodes.

Lamentablement, al comentar aquest emulador amb el director del projecte, ja que semblava el definitiu, veiem que GNS3 és només un emulador de xarxes, no simulador. Això vol dir que es poden emular els nodes i la seva configuració, però el medi físic no es simularà, i, ja que un dels aspectes més importants del projecte és simular el medi sense fils, aquest emulador queda descartat.

## 11. Primeres conclusions

S'han explorat diverses opcions per a fer córrer qMp a CORE, també altres emuladors que podrien arribar a córrer qMp als nodes, però totes les opcions s'han acabat descartant.

Donat que el temps del projecte es limitat, i sabent que segurament es podria arribar a executar qMp als nodes de CORE, s'ha decidit amb el director del projecte deixar aquesta implementació per al final del projecte, si sobra temps, o per a *future work*.

Donat la dificultat tècnica i la manca de documentació, hem decidit gastar el temps en aconseguir emular la xarxa de Guifi.Sants, i avaluar protocols com OSPF, el qual es molt semblant a BMX6, que ja venen per defecte a CORE.

## 12. Xarxa Mesh a CORE

Al bloc de tasques 3 implementarem primer una petita xarxa Mesh sense fils a CORE, per entendre el funcionament de l'emulador més en profunditat, i veure de quines eines disposem per a després poder implementar la nostra xarxa.

Després implementarem la xarxa de GUifi.sants, agafant la informació real de la xarxa i traduint-la al format dels fitxers de configuració d'escenaris de CORE.

## 12.1. Petita xarxa Mesh de prova

La primera prova consta de 9 nodes, els quals repartirem aleatòriament sobre l'escenari de l'emulador, però mantenint una distància prudent per a assegurar-nos de que puguin interconnectar-se. Aquests nodes es deixaran de moment sense configurar cap IP ni opcions addicionals. El protocol OSPF i OSPFv2 venen per defecte activats als nodes.

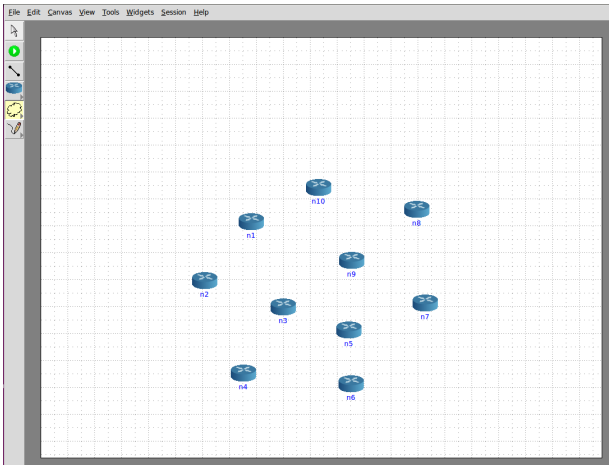



Figura 14: 9 nodes repartits de manera aleatòria a distancia relativa

A continuació es crea un link-layer de tipus WLAN, per a que els nodes es connectin a ell com si fos un ISP. Aquest tipus de link-layer es configurarà amb la IP de subnet desitjada, per a que els nodes puguin vincular-se a la xarxa desitjada.

També disposem d'opcions avançades, com triar el protocol d'EMANE (un emulador de xarxes Ad-hoc), i si triem els protocols de wifi 802.11a/-b/g podem personalitzar elements de la connexió com la distancia màxima del rang, ràtio de transferència, i diverses opcions interessants que ens poden ser útils per a l'escenari de Sants.

Un cop creada la WLAN, si vinculem amb tots els routers amb la opció *Link to all routers*, veurem que automàticament els nodes agafen per DHCP una direcció dins del rang configurat a la WLAN, i podrem veure com apareix una antena a tots els nodes junt amb la seva IP, indicant que el node forma part de una connexió sense fils amb la WLAN, tal i com es mostra a la Figura 10.

Un cop vinculats tots els nodes a la WLAN, amb les seves respectives adreces IP, passem al mode d'execució. Prenem *Play* i veiem que automàticament, després de uns segons de reconeixement, tots els nodes formen una Mesh, creant les adjacències que el protocol OPSF creu més adients per a cada node. (Figura 11)

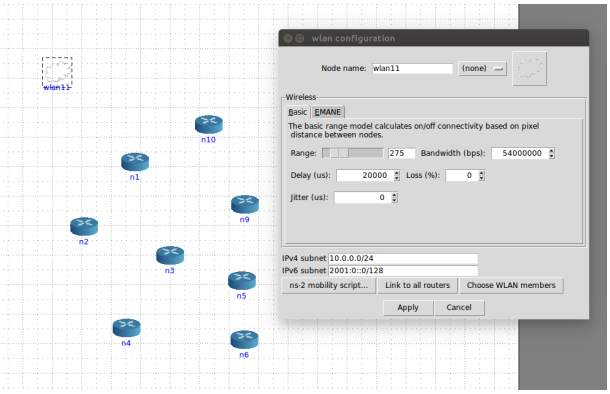



Figura 15: Creació i configuració bàsica de WLAN

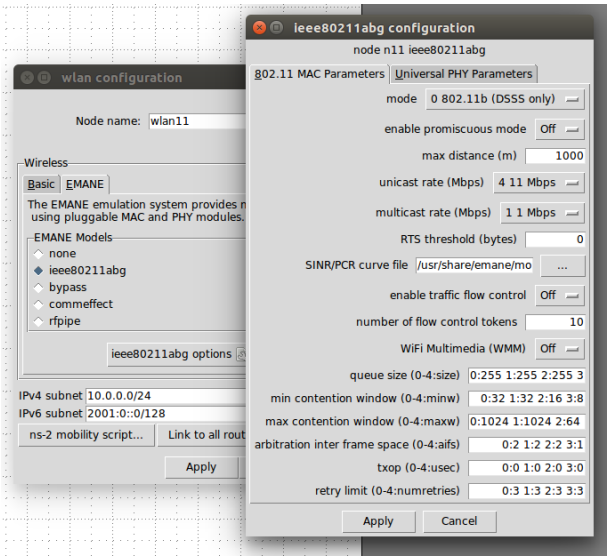



Figura 16: Opcions avançades de la WLAN

Encara en el mode execució, si seleccionem un node i l'arrosseguem de manera que es vagi allunyant de l'escenari, veurem com a mesura que s'allunya del seu estat inicial va format nous enllaços que el protocol creu més adients, fins a arribar a un punt en el que esta tan lluny que perdem la connexió amb els demès nodes. (Figures 12 i 13)

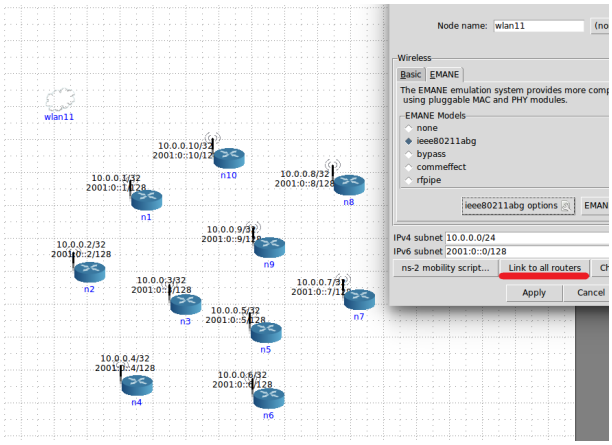



Figura 17: Vinculant tots els nodes a la WLAN principal

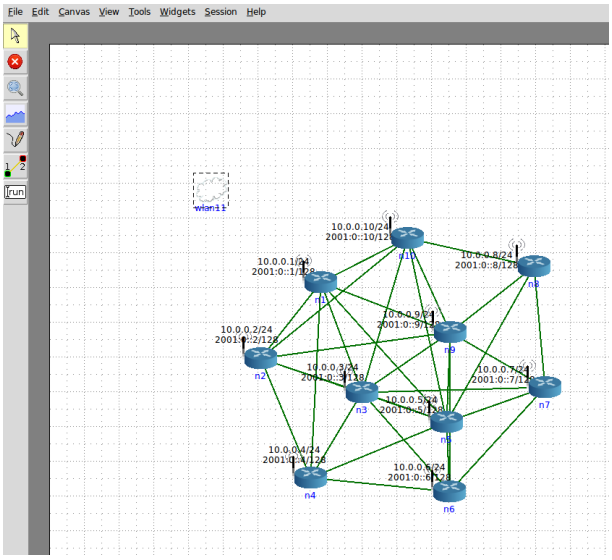



Figura 18: Mode execució: Primera formació de la Mesh

Si guardem l'escenari com a fitxer *.imn*, podem veure la informació que guarda CORE dels nodes de la WLAN, aspecte que ens serà vital per a aprendre el format en el que hem de construir la nostra xarxa de Sants.

Es realitza una petita prova addicional, agafant el fitxer *.imn* generat per CORE al primer escenari en que els nodes no tenen cap IP assignada encara (Figura 7). S'escriu a l'arxiu de configuració per a que els nodes tinguin una interfície a la xarxa 10.0.1.0/24 (per defecte

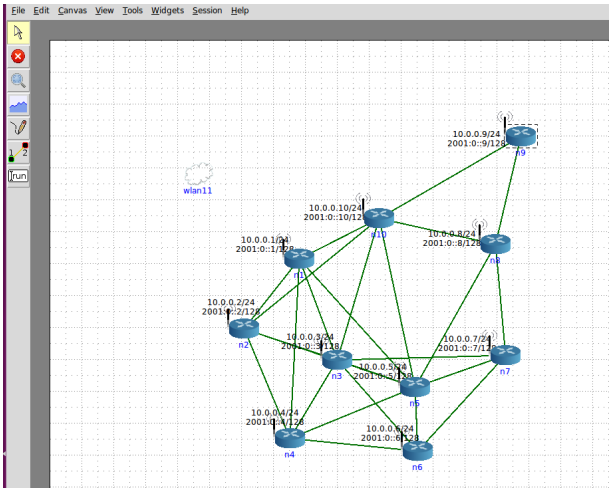



Figura 19: Mode execució: Separant un node de la Mesh

CORE fa servir la 10.0.0.0/32). Es crea la WLAN i es configura amb la xarxa 10.0.1.0/24, i, quan executem l'escenari, veiem que els routers es vinculen amb la WLAN amb les IP's que els hi hem assignat, no per DHCP.

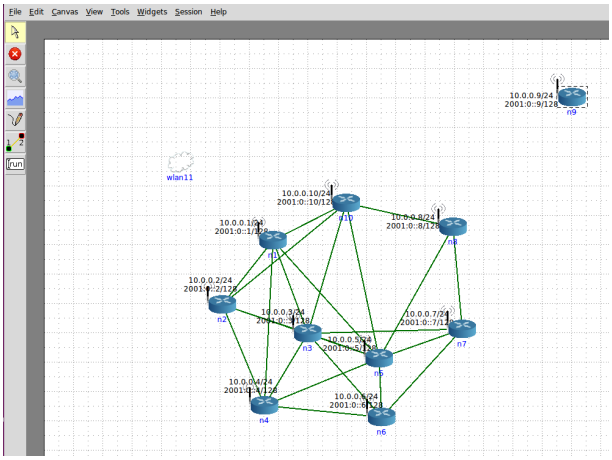



Figura 20: Mode execució: Separant un node fins a sortir del rang

Això vol dir que, si per exemple configurem un node amb la IP 10.0.1.1/24 i existeix una WLAN d'aquesta subnet, al vincular reconeixerà al node amb la IP que li hem assignat, i no agafarà una IP per DHCP del rang per defecte 10.0.0.0/32.

A partir d'aquesta prova podem determinar que, un cop definits tots els nodes, si definim tots els nodes de la xarxa Guifi.Sants, i també definim tots els nodes que actuen com gateway WLAN, amb les IP's corresponents, al fer el primer discover la xarxa hauria de formar els mateixos links que a la xarxa real, o almenys molt aproximat, ja que, apart de no poder fer servir BMX6, l'entorn real sempre constarà de elements que en el simulador no podrem plasmar, com ara bé edificis o elements de l'entorn que de alguna manera modifiquin la senyal wifi.

## 12.2. Creant la xarxa Guifi.Sants a CORE

En aquest apartat finalment començarem a crear la xarxa de Sants a dins de CORE. Per a portar a terme aquesta tasca es necessari conèixer tota la informació possible de la xarxa, la qual el director del projecte ens facilita, i conèixer el format en el que CORE guarda els seus escenaris.

## 12.3. JSON amb informació de la xarxa

La informació de la xarxa Guifi.Sants és pública i es pot descarregar en format JSON de la següent pàgina de monitorització (amb l'enllaç "Download graph"que hi ha en panell de la dreta) (http://dsg.ac.upc.edu/qmpsu/index.php)[15] un fitxer amb format JSON (JavaScript Object Notation, un format de text lleuger per a intercanviar dades) en el qual tenim tota la informació de la xarxa.

Si observem el fitxer (Figura 14), podem veure que mostra informació rellevant de la xarxa, com característiques, canals, i la llista de nodes.

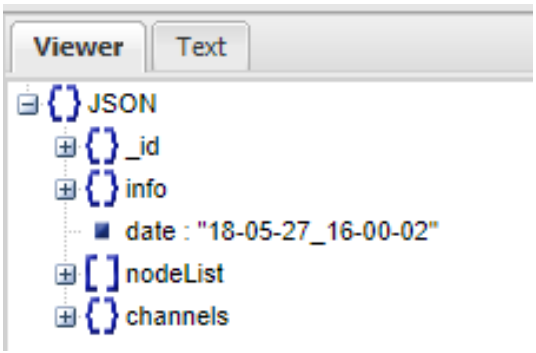



Figura 21: JSON amb la informació de Guifi.Sants

Dins del camp *nodeList*, podem veure tots els nodes que formen la xarxa. El que haurem de fer bàsicament es recórrer aquest array i

convertir-lo al format *.imn* de CORE.

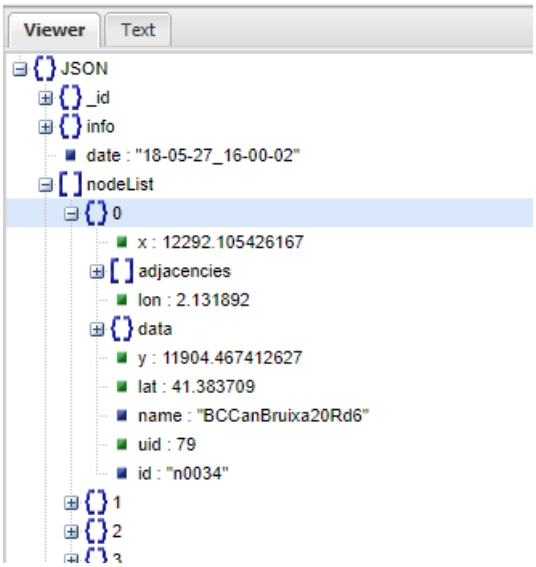



Figura 22: Informació del node 0 dins de nodeList

## 12.4. Convertint de JSON a .imn

Per a realitzar aquesta tasca farem servir Python, donat la seva facilitat a l'hora de gestionar text i cadenes a partir d'un fitxer. Primer es tractarà el string JSON d'entrada, i es recorreran tots els nodes agafant la següent informació:

- Nom del host
- IP
- Posició X
- Posició Y

De moment només necessitem aquestes dades, encara que tampoc es disposen de moltes més, però ja hauria de ser suficient per a l'estudi que volem realitzar. La posició X i Y son les posicions respecte al mapa de Sants. S'intentarà trobar la relació de les coordenades amb Google-Maps per a poder identificar quin es el perímetre exacte de Sants al que es refereix, per a poder agafar el mapa i posa-lo de fons.

De moment, les coordenades reals que s'obtinguin del JSON es dividiran entre 10, ja que d'altre manera el mapa es farà massa petit a CORE, dificultant veure els nodes. Cal dir també que hi han tants

nodes que potser la vista gràfica pot ser una mica incòmode, ja que serà impossible que es vegin totes les etiquetes tant de nom de node com de IP.

## 12.5. Resultat

A continuació podem veure el resultat inicial després de llençar el script de conversió. Cal dir que la xarxa només té els 86 nodes de la llista, no ha tingut en compte si molts dels nodes fan de WLAN i potser s'han de representar com a tals, però de moment ja tenim la topologia de Sants a un llenç de CORE.

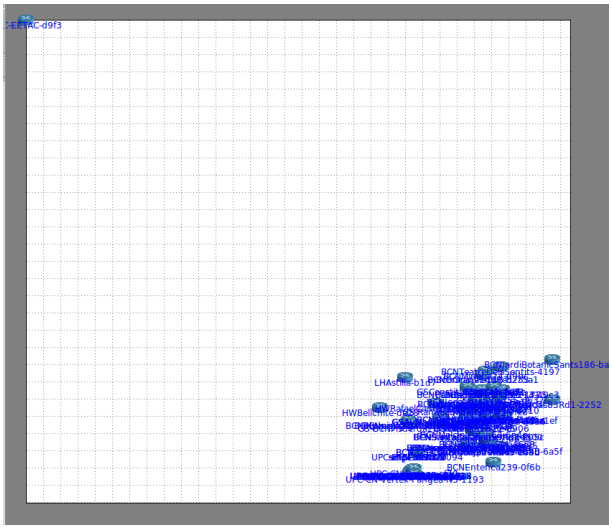


Figura 23: Guifi.Sants a CORE

Com es pot apreciar a la imatge, hi han tants nodes que no es veu molt be, inclús si fem zoom, però ja es normal, donat també que molts nodes es troben exactament a la mateixa ubicació física, o molt a prop els uns dels altres.

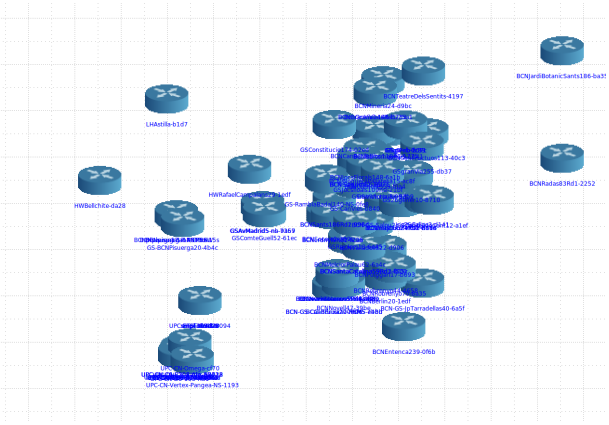


Figura 24: Visió augmentada de la zona de Sants

## 13. Següents passos

El següent serà fer una reunió amb el director del projecte, per a acabar de definir quins tipus de node podem trobar a la xarxa Guifi.Sants, de manera que sapiguem quin son els gateways, i quins dispositius tenen accés a xarxes externes com Internet. Aleshores aquests nodes actuarien com WLANs dins de CORE, i els dispositius es vincularien a aquests nodes.

A continuació, un cop després de tenir tota la xarxa configurada igual que a Sants, es definirà un grup de proves per a avaluar la xarxa, tal i com està definit al bloc de tasques 4.

## 14. Planificació fita de seguiment

Després de realitzar varies proves s'ha trobat que la complexitat per a poder encastar el sistema qMp a CORE, donat la poca documentació, faci pujar molt en temps d'aprenentatge. Per tant, de moment es descarta la opció de qMp a CORE com a *future work*.

Per tant, es passarà a avaluar exhaustivament la xarxa actual de Guifi.Sants amb OSPF i potser algun protocol més, i, si sobrés temps, es seguiria intentant encastar qMp, ja que només seria aconseguir fer córrer dins dels nodes, donat que l'escenari ja el tenim preparat.

Donat aquests canvis podríem dir que s'han assolit un 60 % aproximadament de les tasques definides a la fita inicial, així com també un 60 % del projecte. Ja s'ha definit la xarxa de Guifi.Sants a CORE i s'han entès tots els conceptes de xarxa, també s'ha intentat aconseguir

convert.py

```
import json
from pprint import pprint
import unicodedata

with open('sants.json') as f:
    data = json.load(f)

outfile = open('guifisants.imn','w')

i = 0

for node in data['nodeList']:

    #dades del node
    name = unicodedata.normalize('NFKD', node['name']).encode('ascii','ignore')

    outfile.write("node " + name + " {\n"
    "    type router\n"
    "    model router\n"
    "    network-config {\n"
    "   hostname " + name + "\n"
    "   !\n")

    y = 0

    #recorrer interficies
    for interface in node['data']['ipv4']:


        v4 = unicodedata.normalize('NFKD', node['data']['ipv4'][y]).encode('ascii','ignore')
        outfile.write(
        "  interface eth" + str(y) + "\n"
        "   ip address " + v4 + "\n"
        "  !\n")

        y += 1

    #coordenades /10
    if 'x' in node:
        cordx = node['x']/10
        cordy = node['y']/10

    outfile.write("    }\n"
                  "    canvas c1\n"
                  "    iconcoords {" + str(cordx) + " " + str(cordy) +"}\n"
                  "    labelcoords {" + str(cordx) + " " + str(cordy+30) +"}\n"
                  "}\n\n")


    i+=1



outfile.close()
```

Figura 25: Codi del script de conversió

encastar qMp als nodes sense èxit. Ara farà falta sotmetre l'escenari a proves i analitzar els resultats, i, si dona temps, afegir canvis nous o provar altres protocols semblants a BMX6.

# 15. OSPF a Guifi.Sants

Un cop representat l'escenari de Guifi.Sants al llenç de CORE, haurem d'aplicar un protocol d'encaminament. En aquest cas farem servir

OSPF, ja que ve integrat a Quagga, un software lliure per a routers que implementa diversos protocols com OSPF, RIP i BGP, el servei del qual ja ve creat a CORE.

## 15.1. Entenent OSPF a CORE

Gràcies als exemples d'escenaris dels que disposem a */.core/configs/* podem veure que, per a xarxes sense fils, es fa servir el protocol OSPFv3 per anunciar les xarxes. OSPFv3 es una nova versió del protocol que dona suport a Ipv6, i també implementa OSPF per a xarxes Ad-Hoc, a través del daemon ospfd6. A CORE veiem que es fa servir als exemples per a emular xarxes sense fils Ad-hoc, per tant, aprofitarem aquesta funcionalitat per a la nostra xarxa de Sants.

Es fa una petita prova per a veure realment quantes línies de configuració necessitem per a que OSPFv3 funcioni. Sembla ser que només s'ha de declarar la xarxa que es vol anunciar per la interfície corresponent. També s'ha d'indicar al fitxer de configuració que es farà servir el servei *Quagga*. Si no posem cap configuració per defecte, ens afegirà els defaults de opsf6 (Figura 26). En el nostre cas, com la xarxa es sense fils, s'anunciara per la interfície eth0, tal i com es fa als exemples sense fils de CORE.

```
  services {zebra OSPFv2 OSPFv3MDR vtysh IPForward}
  custom-config {
custom-config-id service:zebra
custom-command zebra
config {
files=('/usr/local/etc/quagga/Quagga.conf', 'quaggaboot.sh', )
}
  }
  custom-config {
custom-config-id service:zebra:/usr/local/etc/quagga/Quagga.conf
custom-command /usr/local/etc/quagga/Quagga.conf
config {
interface eth0
  ip address 10.0.0.5/32
  ipv6 address a::3/128
  ipv6 ospf6 instance-id 65
  ipv6 ospf6 hello-interval 2
  ipv6 ospf6 dead-interval 6
  ipv6 ospf6 retransmit-interval 5
  ipv6 ospf6 network manet-designated-router
  ipv6 ospf6 diffhellos
  ipv6 ospf6 adjacencyconnectivity uniconnected
  ipv6 ospf6 lsafullness mincostlsa
!
```

Figura 26: Configuració d'un node OSPF6 extret dels exemples de CORE

```
router ospf6
  router-id 10.0.0.5
  interface eth0 area 0.0.0.0
  redistribute connected
  redistribute ospf
!
```

Figura 27: Configuració de node OSPF6 que anuncia les xarxes als neighbors

Com es pot veure a la figura 27, el node anunciarà la seva xarxa a través de opsf6 per la interfície eth0, que es la que es farà servir per a la interfície sense fils, encara que sembli que al ser eth0 serà la interfície física, aquesta també s'anunciarà a les connexions sense fils.

Després de fer la prova, s'arriba a la conclusió que només es necessiten les línies de configuració de la figura 28 per a cada node per tal d'anunciar la seva xarxa ospf als seus veïns. En el mode execució, a cada node, disposem dels fitxers de configuració de Quagga a dins de usr.local.etc.quagga/Quagga.conf, que conté les configuracions

que s'estan executant en el node en qüestió. Si observem aquest fitxer després d'aplicar la configuració esmentada anteriorment, podem veure que s'afegeixen línies automàticament amb configuracions per defecte dels protocols opsf i ospf6 (figura 29).

```
  services {zebra OSPFv2 OSPFv3MDR vtysh IPForward}
  custom-config {
config {
interface eth0
  ip address 10.1.9.72/32
!
router ospf6
  router-id 10.1.9.72
  interface eth0 area 0.0.0.0
  redistribute connected
  redistribute ospf
!
}
```

Figura 28: Configuració mínima OSPF6


Figura 29: Quagga.conf dins d'un node en execució

Un cop configurats els nodes, necessitarem que aquests estiguin inter-

connectats entre ells. Per a aconseguir això necessitem un host WLAN que els connecti a tots, com hem vist a seccions anteriors. Per tant farà falta crear aquest host WLAN que connectat a tots els nodes, afegint la connexió individualment a cada node, i després també creant tants links com connexió WLAN-node siguin necessaris.

## 15.2. Prova OSPF a Guifi.Sants

Un cop conegudes les configuracions bàsiques dels nodes per a OSPF, procedim a fer la primera prova del protocol a la xarxa de Guifi.Sants anteriorment creada.

Afegim al nostre script de conversió del JSON de Sants, per a cada node, les configuracions OSPF. Afegirem també el node WLAN connectat a tots els nodes, i creem links virtuals per a tots els nodes amb aquest. El resultat es el de la figura 30. Hem decidit eliminar un node que estava ubicat a Castelldefels, ja que aquest s'ubicava molt lluny dels altres al llenç de CORE i costava de visualitzar, i realment no es determinant per a la simulació que volem representar. D'aquesta manera es molt mes fàcil visualitzar la xarxa de Guifi.Sants al complet al llenç. També s'han reduït les X i Y per a tal de que no quedin espais blancs i es visualitzi correctament.

Per a la WLAN, fa falta definir un rang, aquest rang es l'abast entre un node i l'altre. Per a la primera prova farem servir un rang de 900 metres, ja que si no molts nodes es quedarien penjats.

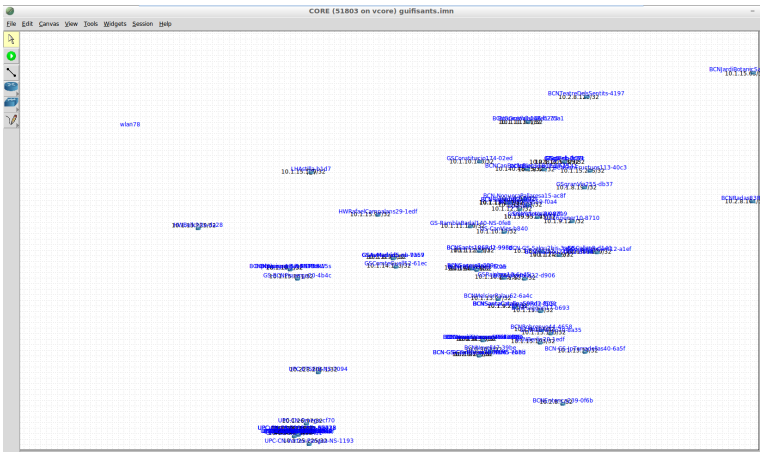



Figura 30: Xarxa de Guifi.Sants

Un cop preparat l'escenari, passem al mode execució. Prenem play i veiem com, després d'uns segons, els enllaços es comencen a crear automàticament entre els nodes, sempre i quan estiguin dins del rang de prova que hem establert (figura 31). Un cop establerts tots els links, veurem com els nodes passen de color vermell a color verd, el que indica que el link es correcte i el node es capaç de detectar al seu veí (figura 32).

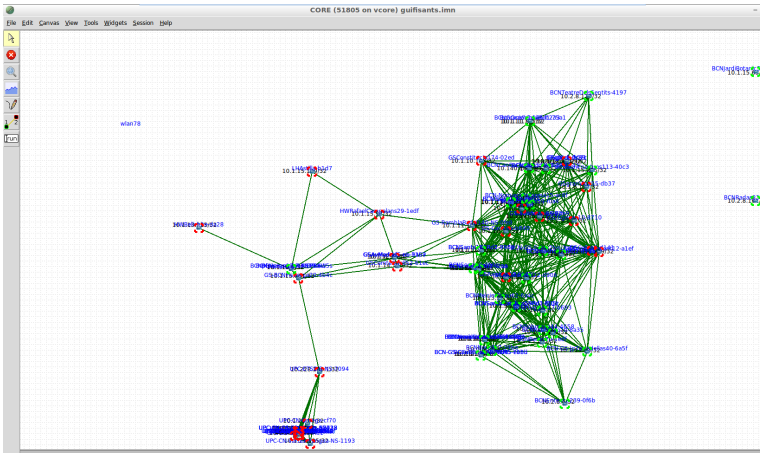


Figura 31: Creació automàtica d'enllaços

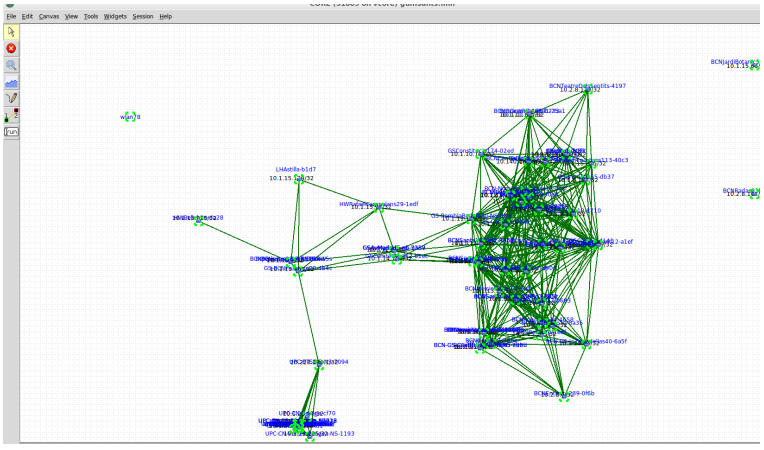


Figura 32: Validació d'enllaços

A continuació, entrarem dins d'un node, i veurem si realment la xarxa OSPF està convergint. Executem *show ip route* a un dels nodes i veiem com, poc a poc, es van coneixent els neighbors, així com les millors rutes OSPF (figura 33).

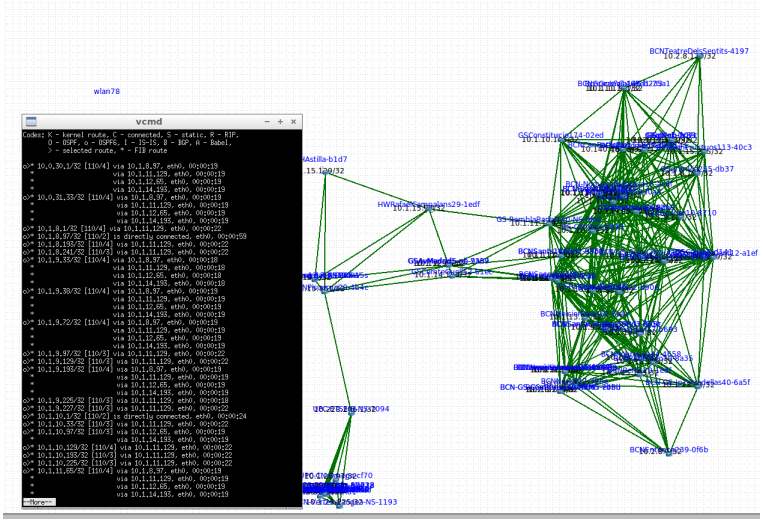



Figura 33: Rutes OSPF a un node

En aquest escenari, i donades les característiques de la màquina virtual que estem executant, la xarxa tarda més o menys 2-3 minuts des de que els links s'estableixen per a convergir la xarxa OSPF al complet. Aquesta dada es aproximada, ja que per a mesurar aquest temps s'observa un dels nodes més allunyats, el qual sempre es un dels que més tarda en agafar les rutes, i s'espera a que tingui tantes línies com neighbors a les rutes. Per a poder mesurar aquesta dada amb exactitud s'hauria de crear un script que preguntés a tots els nodes les rutes, per així veure si realment la xarxa estaria convergida al complet. En aquest apartat, però, no necessitem aquest punt de detall, només necessitem saber si realment OSPF està funcionant correctament, cosa que queda demostrada, ja que al fer ping des de un node a qualsevol altre de la xarxa funciona (figura 34). També podem comprovar que totes les rutes acaben arribant a tots els nodes.

Podem veure que hi han dos nodes que queden despenjats de la xarxa a causa del rang (figura 35). Ens trobem amb el primer problema per a emular el medi d'una xarxa com la de Sants, ja que a la xarxa real cada node pot tenir un rang diferent, i molts nodes i antenes no son omnidireccionals, tal i com emula CORE. També cal esmentar que en el medi real existeixen limitacions físiques com ara bé edificis, o altres senyals que puguin afectar al medi sense fils i que no podem simular a CORE.

Com tots estan connectats a la mateixa WLAN, i el rang sense fils

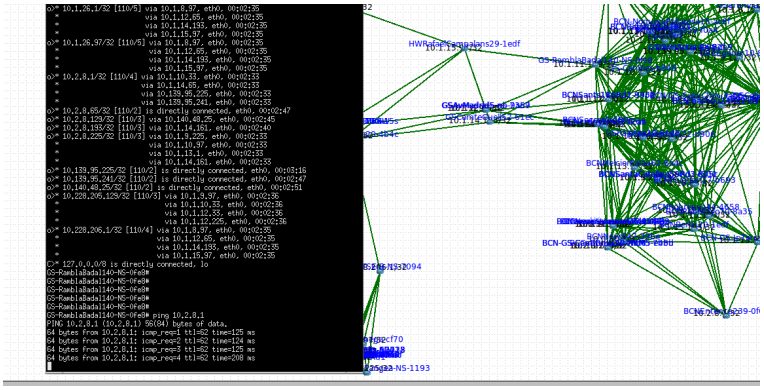


Figura 34: Ping entre nodes

s'ha de configurar a la WLAN, tots els nodes tindran el mateix rang. Fem una prova pujant el rang a 1500 metres (figura 36), i veiem com, un cop tornem a executar la simulació, es creen molts mes links i cap node queda aïllat (figura 37). En aquest cas potser hi haurien masses links, ja que gran part dels nodes son capaços de veure a més d'un 50 % de la xarxa.

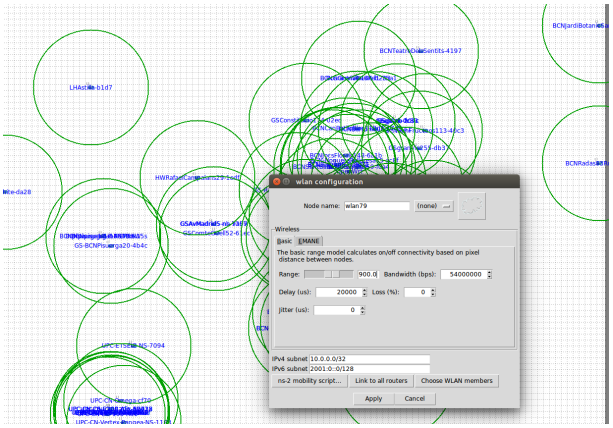



Figura 35: Rang WLAN 900 metres

## 15.3. Conclusions

Hem comprovat que a l'hora de simular la creació d'enllaços sense fils i convergir la xarxa amb OSPF6, la simulació es bastant real. S'aconsegueix que els propis nodes creïn els enllaços i propaguin les seves àrees OSPF, fins a arribar a convergir en un temps que potser no s'ajusta al temps real donat la càrrega de simulació donat el número de nodes, però els protocols es comporten com ho farien a una xarxa real.

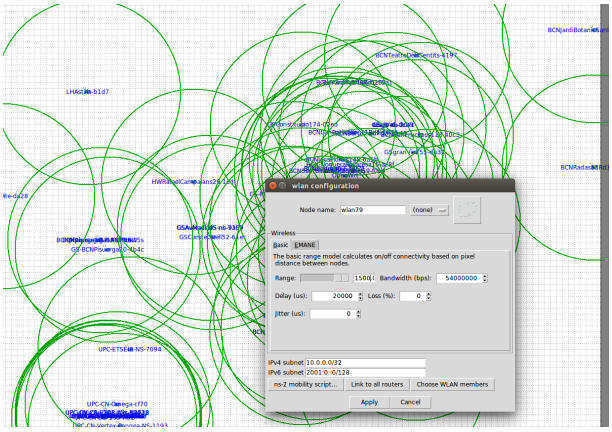



Figura 36: Rang WLAN 1500 metres

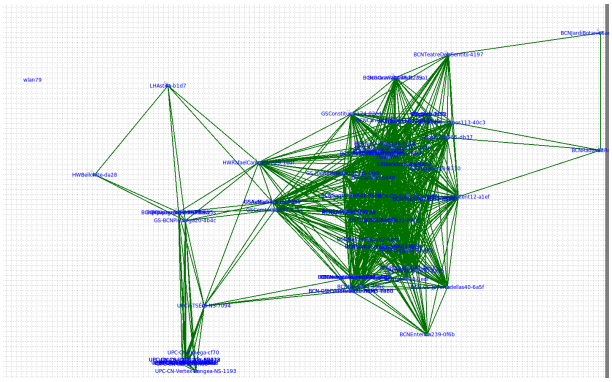


Figura 37: OSPF després d'augmentar l'abast

El problema sorgeix quan volem simular el medi, ja que a un entorn real hi han molts components que poden variar, com edificis, desnivells, etc. que modifiquen en molts casos el comportament de la senyal. Un altre aspecte molt important es que només podem simular senyals de wifi omnidireccional a CORE, mentre que a una xarxa com Guifi.Sants es disposa de moltes antenes unidireccionals, que potencien la senyal projectada cap a un punt en contret.

El fet de no poder configurar l'abast de cada node per separat, lligats al fet que la senyal només pot ser omnidireccional, fan que la xarxa generada a l'execució crei uns enllaços que son diferents als reals de Guifi.Sants. També cal dir que el protocol no es el mateix, ja que estem fer servir OSPF en comptes de BMX6, però a la pràctica son dos protocols molt semblants en termes de millor ruta, i el resultat

s'assimilaria bastant.
Donat que les antenes a CORE son omnidireccionals, podem solventar aquest problema definint els enllaços explícitament al simulador, al següent apartat.

## 15.4. Enllaços estàtics + OSPF

Donades les limitacions descobertes a l'hora de simular el medi, es procedirà a crear la mateixa xarxa però fixant els enllaços igual que estan fixats a la xarxa real de Guifi.Sants. Disposem d'aquesta informació al mateix JSON que fem servir, per tant només haurem de veure com la podem representar a CORE.

Podríem crear un enllaç físic a l'emulador per cada enllaç entre nodes real de la xarxa, però aleshores ja no estaríem en mode sense fils i l'escenari perdria sentit. Per tant, crearem una WLAN per a cada enllaç, vinculant els dos nodes amb aquesta i creant el link corresponent al fitxer de configuració.

Un punt fort d'aquesta simulació és el fet que puguem establir l'ample de banda de cada enllaç segons la informació d'aquest al JSON, de manera que la velocitat serà la mateixa a l'emulador. Per tant, al script, a l'hora de crear cada node, agafarem les seves adjacències i crearem tantes interfícies de xarxa com enllaços tingui (figura 38), i els referenciarem al node WLAN en qüestió (figura 39), després al node WLAN (figura 40) també farem referencia als nodes connectats (figura 41) i es crearan 2 links, un per cada node-wlan (figura 42).

```
node n36 {
    type router
    model router
    network-config {
  hostname BCN-GS-JpTarradellas40-6a5f
  !
  interface eth0
   ip address 10.1.13.33/32
  !
  interface eth1
   ip address 10.1.13.33/32
  !
  interface eth2
   ip address 10.1.13.33/32
  !
  interface eth3
   ip address 10.1.13.33/32
  !
  interface eth4
   ip address 10.1.13.33/32
  !
```

Figura 38: Tantes interfícies de xarxa com enllaços

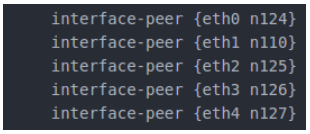



Figura 39: Connexió amb les WLANS

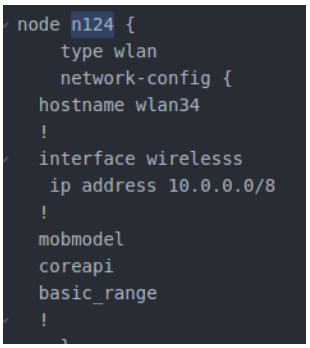



Figura 40: WLAN124 que fan de enllaç entre 2 nodes

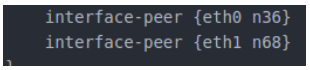



Figura 41: Enllaços WLAN124

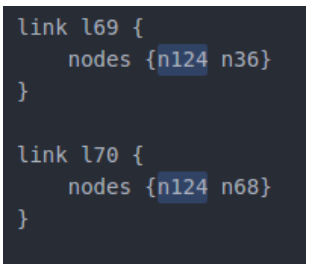



Figura 42: Links WLAN124 amb nodes

Un cop creada la configuració, després d'haver de respectar una sèrie de regles en quant a numeració de nodes i més aspectes que es comentaran més endavant, s'executa el fitxer a l'emulador. Les icones de les wlans s'han disposat a dalt del llenç de manera que es pugui veure el seu estat (figura 43). Si alguna wlan està vermella a la fase de inicialització, algun dels enllaços no està funcionant correctament (figura 44).

Es tarda aproximadament 1 minut en inicialitzar tots els enllaços. Un cop tots validats començarà la convergència de OSPF. Es pot observar, amb una de les eines de previsualització que ens facilita el mode

```
bw = str(int(float(data[2])*1000000))
outfile.write("node n" + str(y+90) + " {\n"
              "    type wlan\n"
              "    network-config {\n"
              " hostname wlan" + str(y) + "\n"
              " !\n"
              " interface wirelesss\n"
              "  ip address 10.0.0.0/8\n"
              " !\n"
              " mobmodel\n"
              " coreapi\n"
              " basic_range\n"
              " !\n"
              "    }\n"
              "    custom-config {\n"
              " custom-config-id basic_range\n"
              " custom-command {3 3 9 9 9}\n"
              " config {\n"
              " range=150000.0\n"
              " bandwidth=" + bw + "\n"
              " jitter=0\n"
```

Figura 43: Escenari de Guifi.Sants

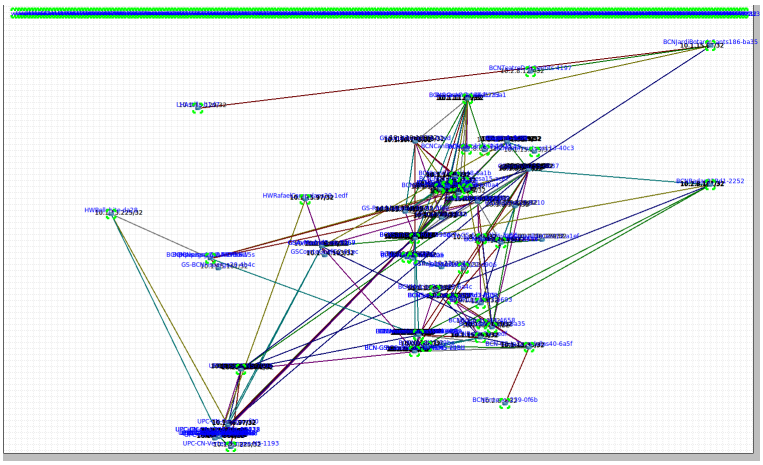


Figura 44: Creació d'enllaços en mode execució

d'execució, com poc a poc es van establint les rutes (figura 45). Cal dir però que el temps de propagació és extremadament lent, a la majoria de casos s'ha arribat a tardar de 15 a 30 minuts en tenir totes les rutes (figura 46), cosa molt estranya considerant que el temps real de convergència a una xarxa com la de Guifi.Sants pot arribar a ser de segons.

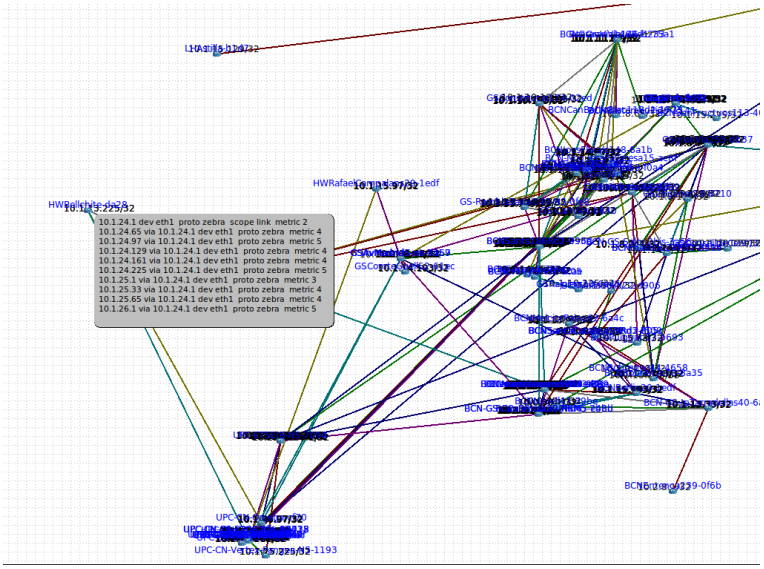


Figura 45: Primeres rutes OSPF

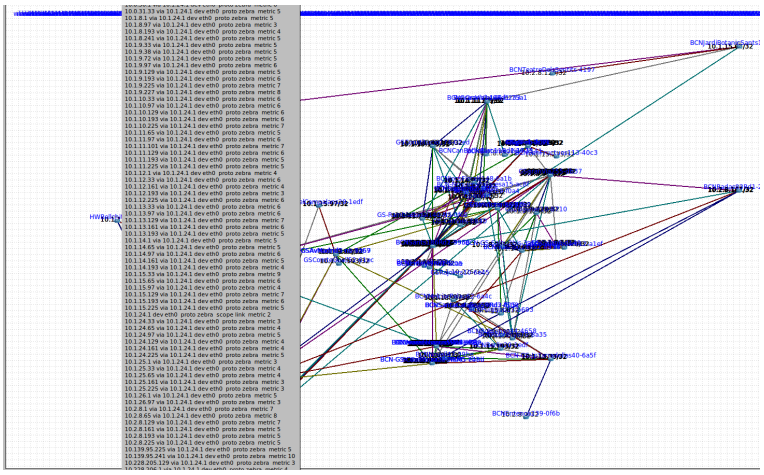


Figura 46: Convergència OSPF

El primer que ens ve al cap es que possiblement, al haver de crear tantes wlans com enllaços, estem sobrecarregant el sistema. Cada wlan és realment un contenidor emulant un node sense fils, per tant cada cop que es vulgui intercanviar informació s'haurà de passar per aquest contenidor, i, al tenir uns 220-230, això fa que el procès s'alenteixi exponencialment a la hora de comunicar les rutes i trobar la més eficient.

Es però una mica estrany que, mentre es propaguen les rutes, l'indicador de CPU a CORE es manté a uns percentatges d'ús molt reduïts, entre

el 1 i el 10 %, en canvi quan estableix els links per primer cop puja fins al 100 %. Podríem assumir que el sistema està sobrecarregat per culpa del que comentàvem anteriorment, però aleshores la CPU hauria d'estar en un percentatge més elevat. Es molt possible que ens trobem amb un problema de *overhead*, ja que al haver de passar per tants contenidors els paquets han de fer masses voltes i acaba alentint el procés bàsic de comunicació entre nodes.

# 16. Simulació i avaluació de resultats

Un cop representada la xarxa sense fils de Guifi.Sants a l'emulador de les dues maneres possibles, procedirem a avaluar el seu comportament i si realment es una eina que ens pot ser útil per a estudis futurs.

Les proves més rellevants son:

- Convergència de OSPF
- Temps de convergència OSPF
- Throughput de la xarxa

Analitzarem el comportament d'aquests aspectes en els dos escenaris representats anteriorment. El primer escenari estableix les connexions sense fils automàticament segons la simulació de CORE, suposant que tots els nodes son omnidireccionals. El segon es fa amb els enllaços ja creats exactament igual que a la xarxa de Guifi.Sants, havent de simular un contenidor per cada enllaç entre nodes.

## 16.1. Convergència OSPF amb enllaços automàtics

Per a poder mesurar el temps de convergència de OSPF, hem creat un script que corri a tots els nodes, el qual cada 10 segons contarà el número de línies que té a la taula d'encaminament. Quan aquest número de línies sigui igual al número de nodes - 1 (el propi node), enviarem una senyal a través de TCP a un host determinat, el qual estarà escoltant per un port per a recollir les senyals.

Per a realitzar aquesta configuració s'ha necessitat ajuda dels exemples que ens facilita CORE, en el qual es crea un script personalitzat, i es posa el codi dins de la declaració del node (figura 47). S'ha de tenir especial cura amb els espais a l'hora de crear el fitxer de configuració, qualsevol espai mal posat o alineació incorrecte pot fer que el node no funcioni correctament i, en conseqüència, l'escenari de simulació.

```
    custom-config {
    custom-config-id service:UserDefined
    custom-command UserDefined
    config {
    files=('mgen.sh', )
    startidx=35
    cmdup=('sh mgen.sh', )
    }
    }
    custom-config {
    custom-config-id service:UserDefined:mgen.sh
    custom-command mgen.sh
    config {
    #!/bin/sh
  uptime | cut -d" " -f2 >> outfile.txt
  result="$(ifconfig | grep 10.1.15.129)"
  if [ "$result" != "" ]; then
   nc -k -l 2399 >> outfile.txt &
  fi
  routes="$(ip r | grep zebra | wc -l)"
  while [ $routes -lt 75 ]
  do
   sleep 10
   routes="$(ip r | grep zebra | wc -l)"
  done
  uptime | cut -d -f2 | nc 10.1.15.129 2399
    }
    }
    services {zebra OSPFv2 OSPFv3MDR vtysh UserDefined IPForward}
}
```

Figura 47: Script a dins de la configuració de tots els nodes

Tal i com es pot veure a la figura 47, el script executarà 'mgen.sh', el codi del qual posem a dins de la configuració dels nodes. Per tant, el script s'executarà a tots els nodes, i només si el node es el 10.1.15.129, activarem el servei *netcat* per a escoltar les senyals que ens enviïn els altres nodes.

S'inicia la execució i s'espera a que les rutes vagin arribant (figura 48). El fitxer que rep les dades amb *netcat* marca l'hora d'inici del script, i els demès nodes aniran enviant l'hora quan tinguin totes les rutes a la seva base de dades d'encaminament. Donades les limitacions a l'hora de segons quines coses amb els scripts bash que s'executen a dins dels nodes, com ara bé que la variables d'entorn $SECONDS i algunes bàsiques no funcionen (retornen null), ja que no s'està executant cap shell, s'agafarà el valor de la comanda *uptime*.

Les comandes que s'executin als nodes agafaran realment els valors de la màquina host, per tant l'hora exacta del host ja ens pot servir de referència, donat que també volem saber un valor aproximat, per tant no fa falta entrar tant en detall en agafar el temps exacte. També tenim en compte que al ser una emulació a una màquina virtual com la que estem fent servir, tot anirà més lent.

Deixem l'emulador encès més de 30 minuts per a assegurar-nos que totes les rutes hagin arribat (figura 49), comptem les línies del fitxer

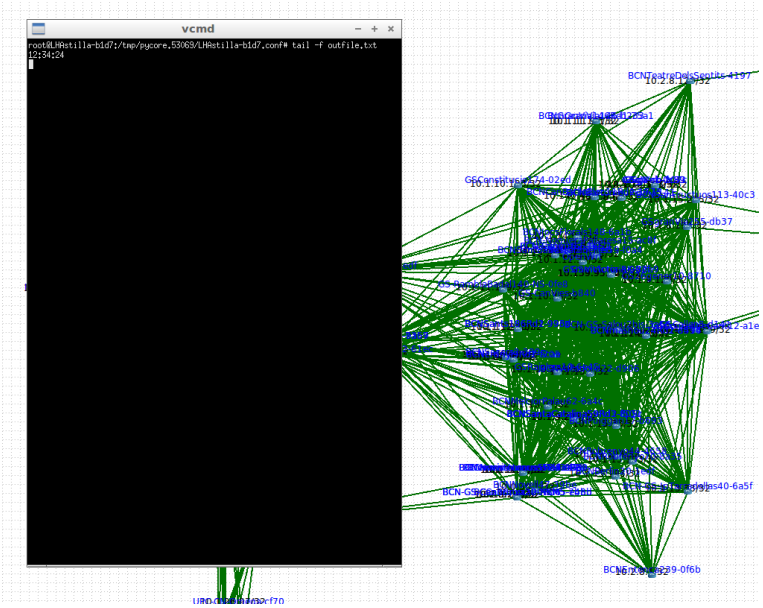



Figura 48: Esperant primeres convergències OSPF

per assegurar-nos que han arribat tants missatges com nodes -1, i restem l'última hora que ha arribat amb la primera del fitxer per a mesurar el temps.

Després d'executar unes quantes simulacions iguals, veiem que hi ha algun punt en el que falla quelcom, en respecte al número de nodes que rebem. En teoria son 80 nodes, menys 1 que està molt lluny (node de Castelldefels), per tant, haurien d'arribar com a mínim 78 rutes, en el millor cas 79. Però per molt que esperem, si anem node per node, podem veure que a la seva taula d'encaminament aquests tenen 75 entrades (figura 50). Per tant hi ha de haver-hi algun problema amb aquests 3-4 nodes que falten, però al haver-hi quasi tots almenys podem calcular quant temps tarda en convergir els 75 nodes.

Es realitzen 5 proves, i es veu que el temps de mitja son 2-3 minuts entre tots els temps aconseguir totes les rutes. Cal dir que no son les 79 que haurien de ser, però es possible que al convergir tota la xarxa de cop, al enviar la senyal tots els nodes, s'hagi perdut alguna perquè el node estava rebent masses dades de cop. En alguns casos el temps supera els 4 minuts, però un altre cop ens podem estar trobant amb aspectes externs que poden afectar a la simulació com per exemple quan l'ordinador està fent servir més recursos fora de la màquina virtual en aquell moment, i fa que aquesta vagi més lenta.

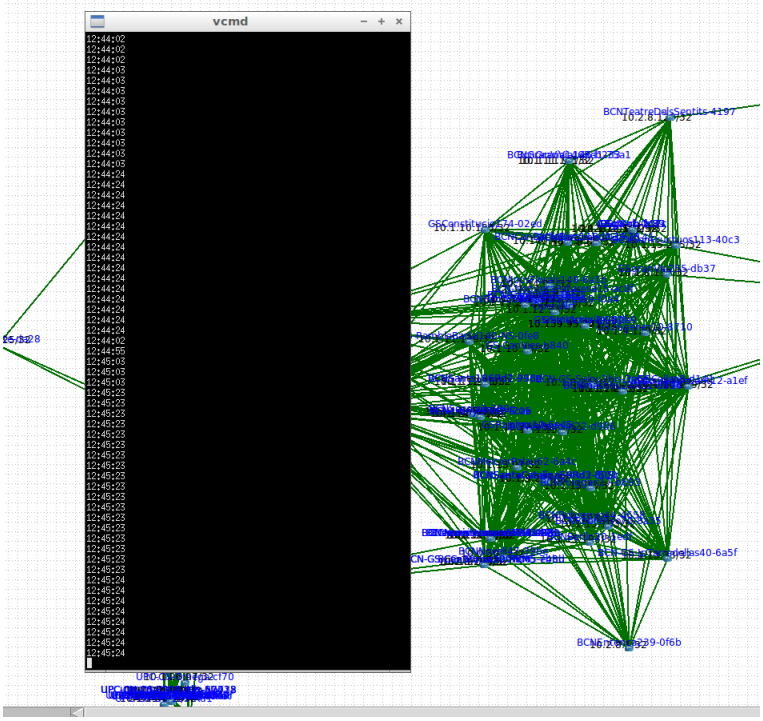



Figura 49: Missatges de convergència OSPF

Es realitzen proves també, en comptes d'esperar a tenir totes les rutes a cada node, fent ping en quant el tinguem disponible el node que fa de servidor (figura 51). D'aquesta manera es veu quan tots els nodes poden arribar al node. Potser no ens indica exactament la convergència completa de la xarxa, però es una bona activitat per a veure quants acaben arribant a la màquina. Un altre cop, els registres que tenim no superen les 60 línies (en almenys 10 minuts). Els temps es mantenen també entre els 2-3 minuts, cosa normal ja que en quant es tenen els paths més curts es quan ospf comença a convergir, per tant molts nodes adquiriran la taula complerta de cop.

## 17. Convergència OSPF amb enllaços estàtics

Seguim exactament el mateix procés que l'anterior però en aquest cas fent servir la xarxa de Guifi.Sants amb els enllaços ja assignats, tal i com els tenim a la xarxa real. Una observació d'aquesta prova es que hem afegit el node de Castelldefels, apropant-lo una mica al canvas, d'aquesta manera també podem veure com estan connectats els nodes de la xarxa real. En tot moment es pot diferenciar que hi han nodes direccionals

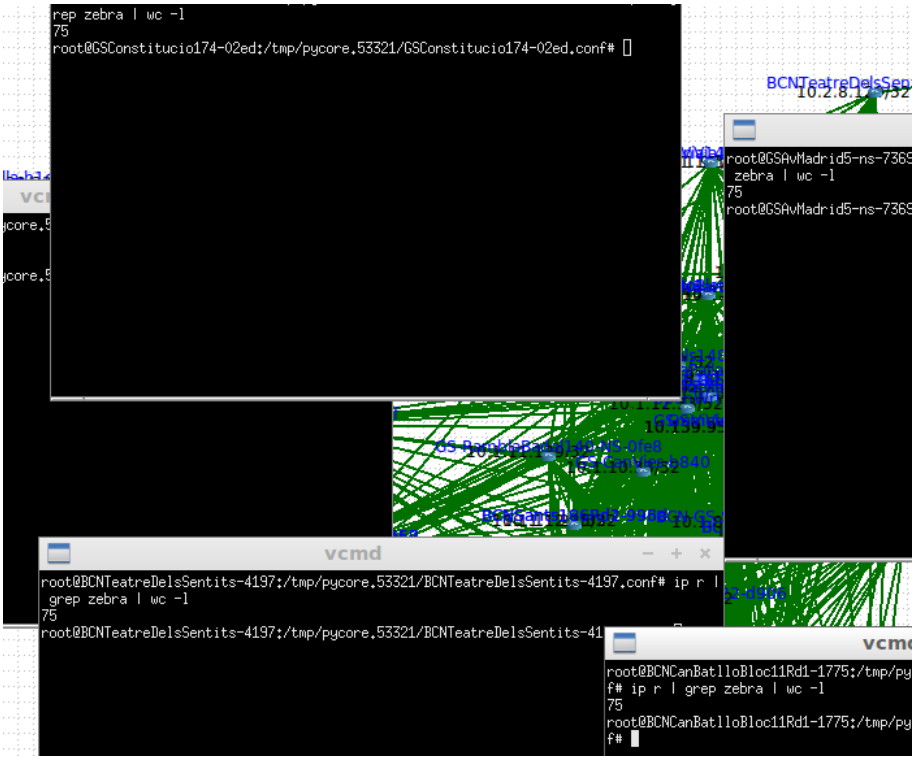



Figura 50: Número de línies a la taula d'encaminament dels nodes

```
"    custom-config {\n"
" custom-config-id service:UserDefined:mgen.sh\n"
" custom-command mgen.sh\n"
" config {\n"
" #!/bin/sh\n"
'  uptime | cut -d" " -f2 >> outfile.txt\n'
'  result="$(ifconfig | grep 10.1.15.129)"\n'
'  if [ "$result" != "" ]; then\n'
'   nc -k -l 2399 >> outfile.txt & \n'
'  fi\n'
"  until nc -vzw 2 10.1.15.129 2399; do sleep 2; done\n"
'  uptime | cut -d" " -f2 | nc 10.1.15.129 2399\n'
" }\n"
```

Figura 51: Script per fer ping al node servidor tan d'hora com es pugui fer

que apunten cap a un lloc determinat, i altres que formen parts de xarxes mes petites omnidireccionals (figura 52).

En aquesta prova, tal i com hem observat anteriorment quan la vem crear, la velocitat de propagació de la taula d'encaminament és extremadament lenta. La primera prova amb el script de convergència ja tarda uns 25 minuts, per això ja no farem les següents proves. No és normal que la xarxa tardi tant en convergir. Queda clar que quelcom

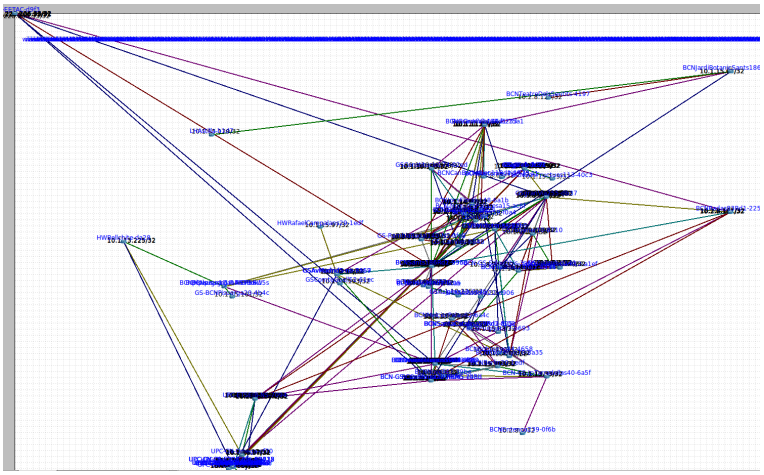


Figura 52: Xarxa amb links estàtics i node Castelldefels apropat

està passant que fa anar a la xarxa massa lenta, segurament causat per el número de contenidors. Només fa falta veure tots els processos que es creen a la màquina host (figures 53 i 54) si executem *ps aux*. Cal comentar però, que tant a la versiò estàtica com a la dinàmica el número de processos creats son els mateixos, però al tenir 225 wlans, una per cada enllaç, tot va molt més lent a la versió estàtica.

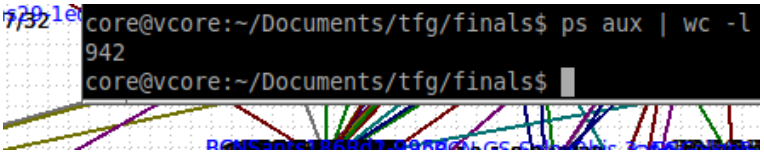



Figura 53: Número de processos executant-se

Per tant, descartem fer més proves en aquest entorn donada la lentitud. Potser amb una màquina virtual més potent, o si s'aconseguís configurar correctament CORE a una màquina amb Linux i que fos potent, es podrien aconseguir altres resultats. Seria interessant com a future work investigar si això seria possible, ja que aquest escenari es l'ideal per a nosaltres per a portar a terme simulacions de protocols a la xarxa. Donat que els enllaços son idèntics a la xarxa real, només ens hauríem de preocupar de simular correctament els protocols.

## 17.1. Throughput

Un altre aspecte molt important, si cap potser el més important després de saber que podem convergir la xarxa amb algun protocol d'encaminament,

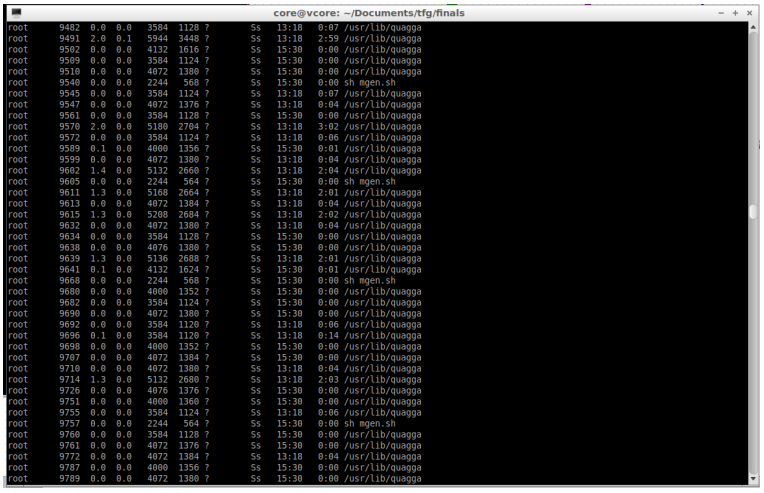



Figura 54: Processos executant-se al node host

és poder mesurar el trhoughput de la xarxa (volum de dades que podem moure per la xarxa). En el primer escenari tenim un inconvenient, i esque al estar tots els nodes connectats a la mateixa wlan, i al només poder definir-se l'abast de senyal des de la wlan, tots estan limitats a tenir el mateix ample de banda.

Disposem d'una eina a CORE que ens mostra el throughput de cada node (figura 55). Els números costen una miqueta de veure amb tants nodes i enllaços però es poden apreciar. Aquest throughput és l'actual entre totes les connexions del mateix node. Quan l'executem podem veure que l'emulador es posa a 100 % i les coses comencen a anar més lentes, degut a que ha de calcular el throughput de 80 nodes. Per a calcular la velocitat de transmissió entre dos nodes, farem servir l'eina de terminal iperf3, amb la qual establim una connexió TCP i simplement intercanviem dades.

Totes les proves es faran 2 cops, una entre 2 nodes adjacents, i l'altre entre dos nodes que estiguin el més allunyats possible l'un de l'altre, de manera que hagin de passar per múltiples nodes per establir connexió entre ells.

Es realitza una prova entre dos nodes adjacents. La wlan està configurada per a tenir links de 56Mbps i un retard de 20 mil·lisegons, però els resultats no passen de 150Kb/s, 500Kb/s en algun pic (figura 56). La velocitat hauria de ser de 54Mbp/s, així que provarem de tocar alguns paràmetres per a veure si aconseguim simular l'enllaç correctament.

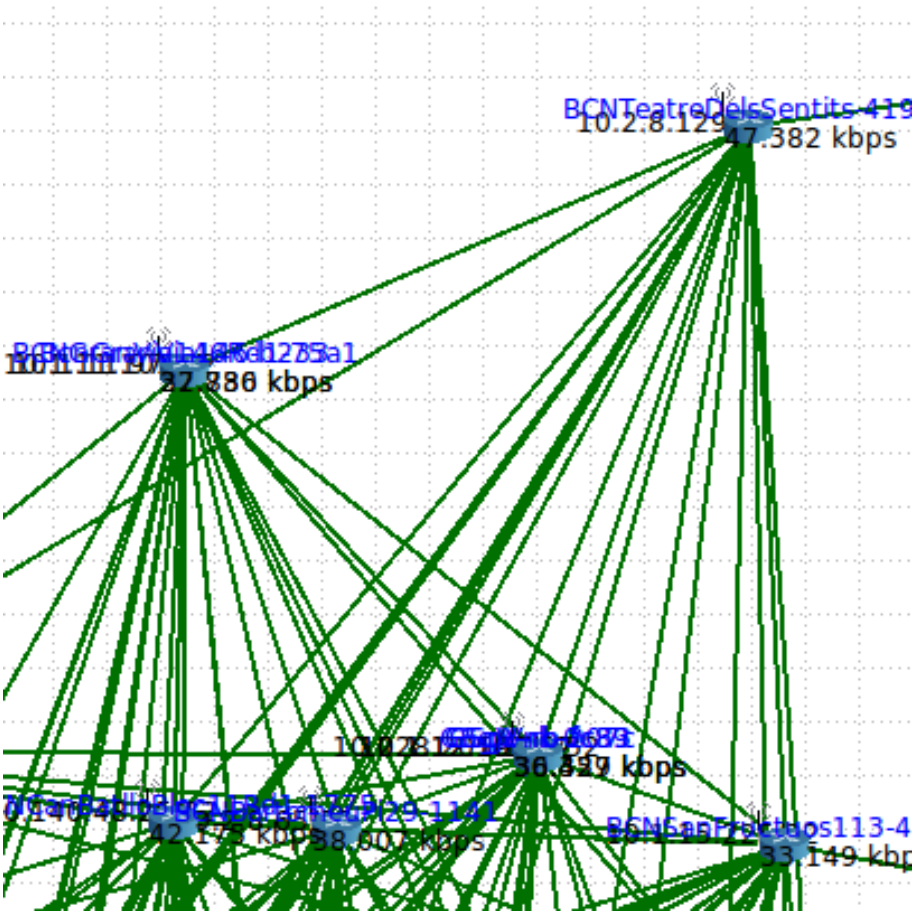



Figura 55: Throughput en directe de cada node

Canviem el retard de la mateixa wlan a 10 mil·lisegons, però veiem que la velocitat és quasi bé la mateixa, es podria dir que les velocitats son molt iguals a la prova anterior. Baixem aleshores el retard dràsticament a 10 nanosegons, i podem observar com la velocitat comença a elevar-se fins a arribar a pics de 40Mb/s als nodes adjacents i 6-7Mb/s als nodes allunyats (figures 57 i 58)

Modifiquem la velocitat de transmissió de la wlan, i la fixem a 200Mb/s, per a veure com es comporta. Veiem que la velocitat augmenta considerablement, en molts casos arriba als 70-80Mb/s en els nodes adjacents (figura 59) i 7Mb/s als nodes allunyats (figura 60), però a vegades es una mica inestable i en alguna prova poden caure els valors, creiem que degut al cost de simulació sumat a la virtualització del sistema. No

augmentarem més la velocitat, donat que ens trobem amb un límit de 80Mb/s quan seguim pujant el bandwith, com hem pogut veure a una prova simple amb pocs nodes que es comenta en els apartats següents.

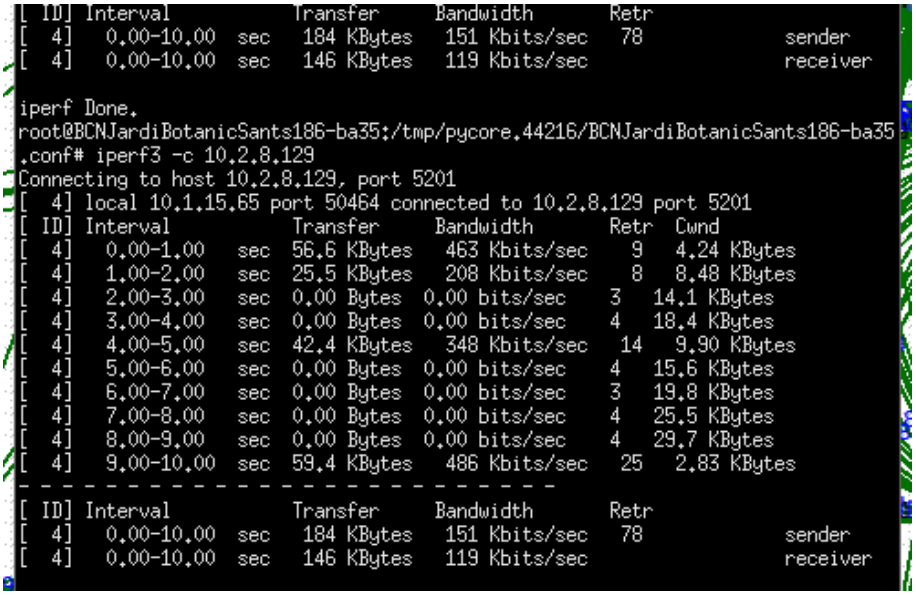



Figura 56: 54Mbps i 20 mil·lisegons de retard nodes adjacents

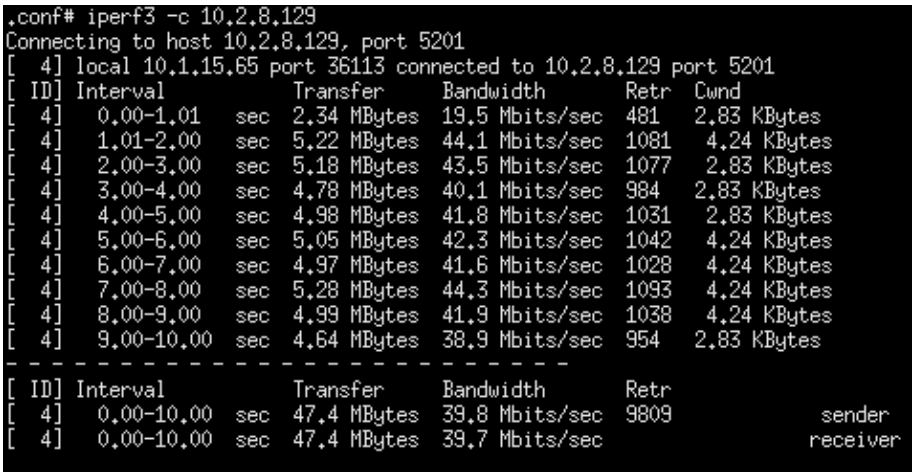



Figura 57: 54Mbps i 10 nanosegons de retard nodes adjacents

Provem aleshores, de posar la velocitat a il·limitada, per a veure com reacciona la xarxa. Això es possible posant l'ample de banda de la xarxa a "0"(figura 61) per a veure com reacciona. Es fa la mateixa

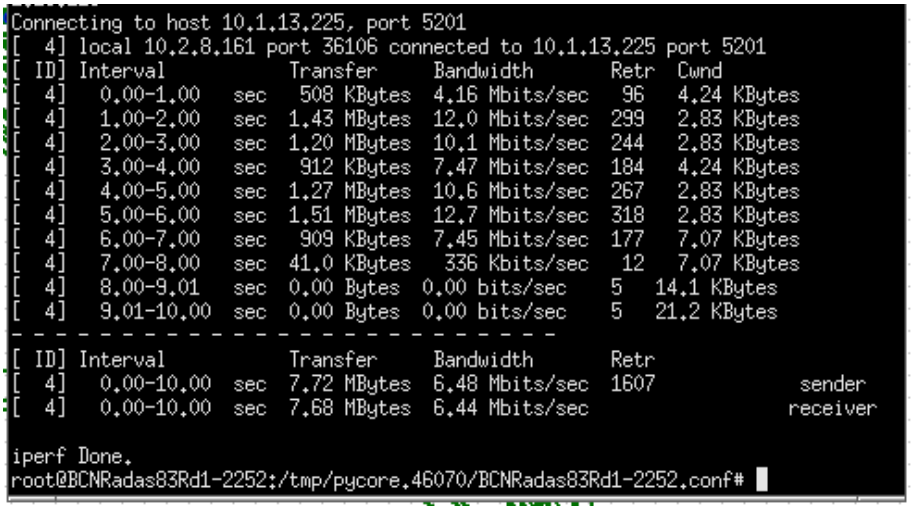



Figura 58: 54Mbps i 10 nanosegons de retard nodes separats

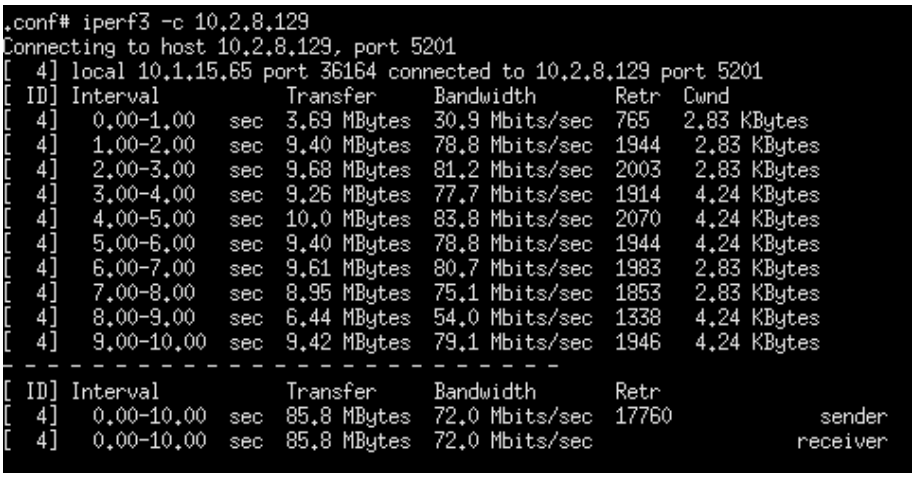



Figura 59: 200Mbps i 10 nanosegons de retard nodes adjacents

prova i veiem que les velocitats augmenten fins a arribar a 3-4Gb/s a nodes adjacents (figura 62) i en alguns casos 1Gb/s en nodes allunyats (figura 63). sorprenentment, la xarxa pot arribar a throughputs molt alts quan no té límit, això fa dubtar una mica dels enllaços de CORE, per això procedirem a fer una petita avaluació a continuació a un entorn simple.

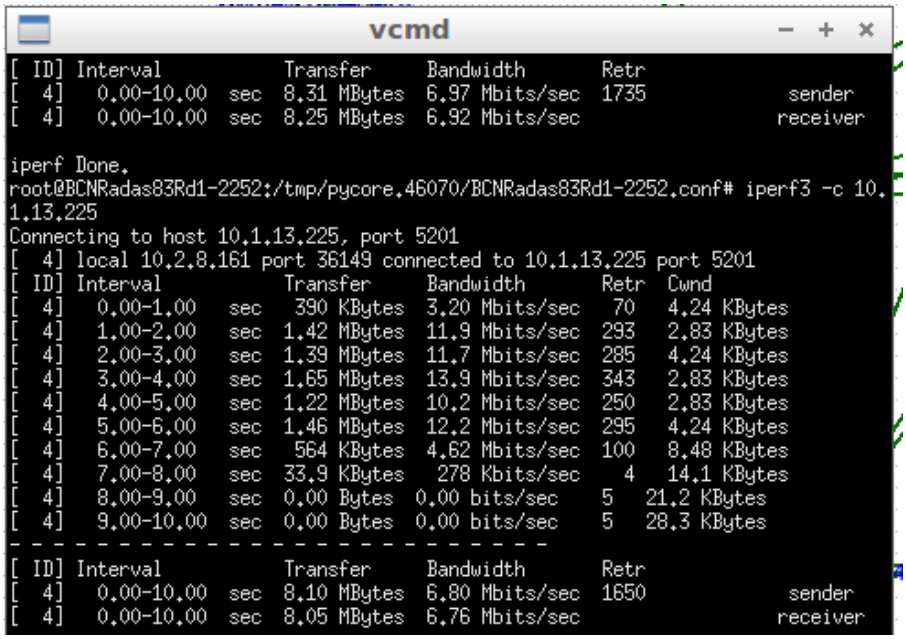



Figura 60: 200Mbps i 10 nanosegons de retard nodes allunyats

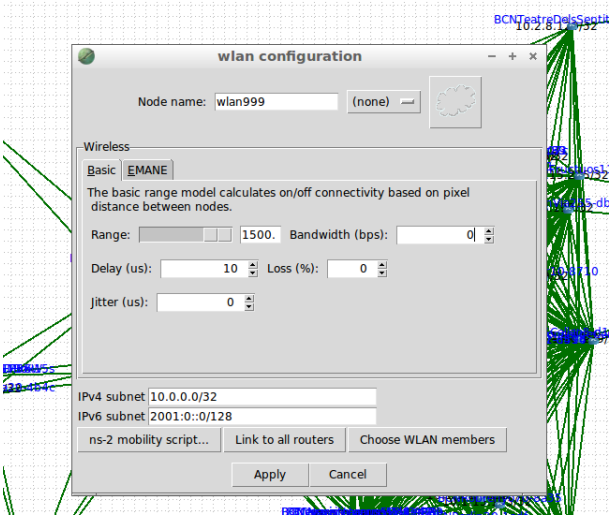



Figura 61: Posem el bandwith en mode il·limitat

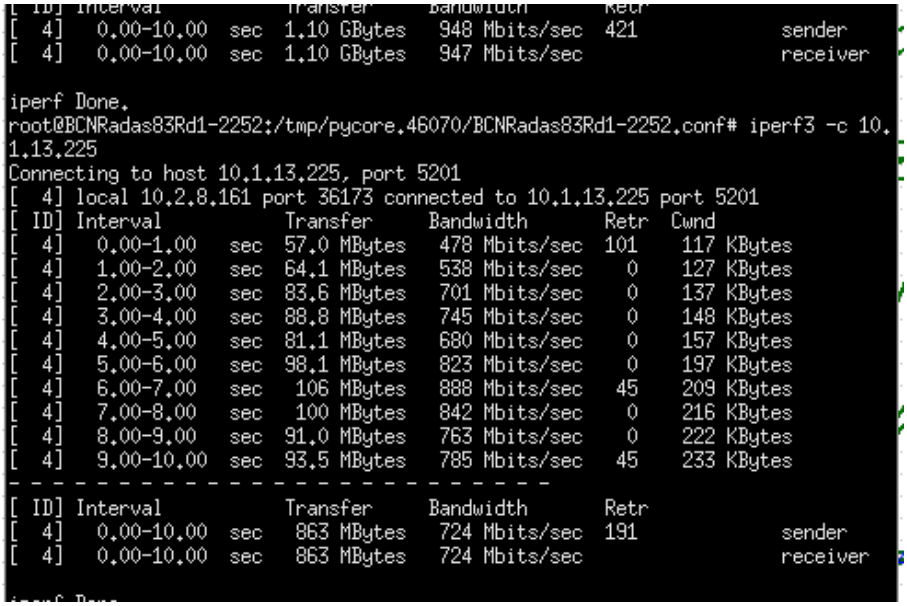



Figura 62: Velocitat il·limitada i 10 nanosegons de retard nodes allunyats

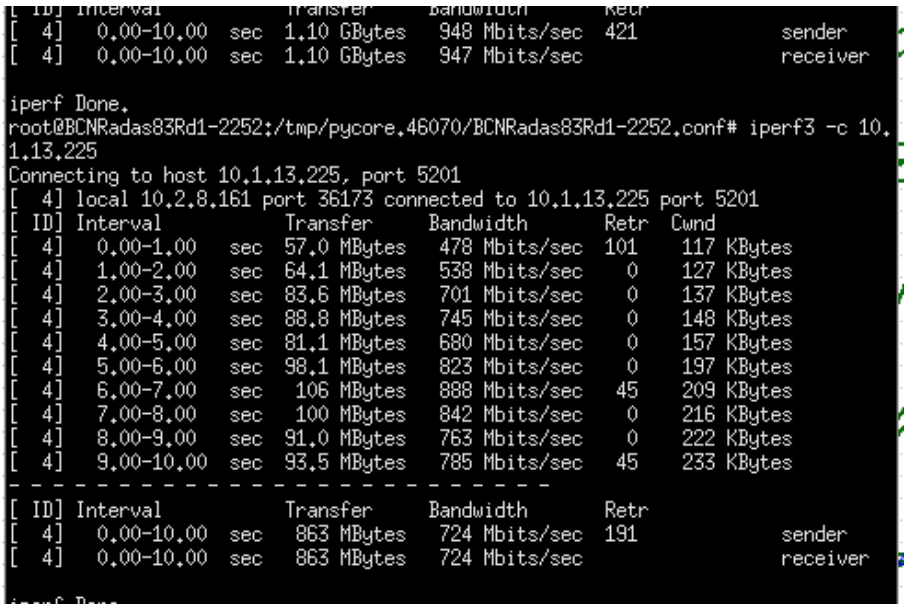



Figura 63: Velocitat il·limitada i 10 nanosegons de retard nodes allunyats

## 17.2. Prova throughput en un entorn bàsic

Donat els comportament dels enllaços a vegades no es l'esperat, realitzarem una prova en un àmbit bàsic amb 2 nodes (figura 64), per a assegurarnos de que no està passant per culpa de tenir molts nodes corrent a la vegada, i de pas conèixer més a fons com funciona aquesta característica a CORE.

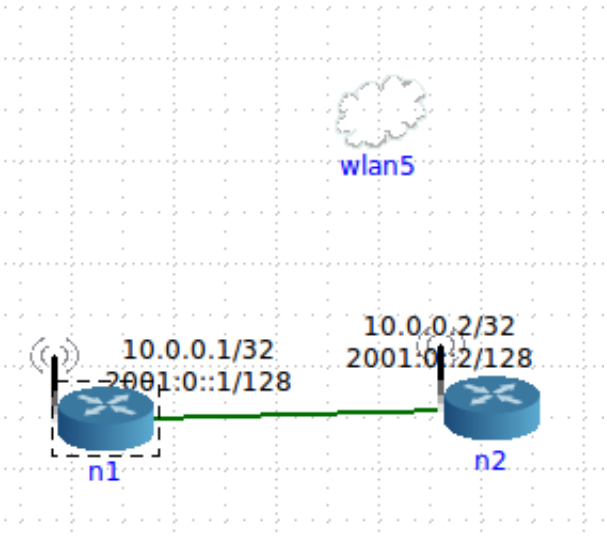



Figura 64: Velocitat il·limitada i 10 nanosegons de retard nodes allunyats

Es realitzen les segones proves, establint connexions TCP entre els dos únics nodes de la xarxa:

- 54Mb/s ample de banda i 20000 nanosegons de retard (figura 65)
- 54Mb/s ample de banda i 10 nanosegons de retard (figura 66)
- 80Mb/s ample de banda i 10 nanosegons de retard (figura 67)
- 300Mb/s ample de banda i 10 nanosegons de retard (figura 68)
- Velocitat il·limitada i 10 nanosegons de retard (figura 69)

Podem veure que, en quant baixem el retard a 10 nanosegons, tot comença a funcionar com esperem, almenys fins a cert punts. Veiem una limitació de la velocitat en quant es passa de 100Mb/s, si posem més de 100, la velocitat es quedarà estancada en uns 100-120Mb/s. En canvi, quan aquesta es posa ilimitada, pot arribar a 10Gb/s. Costa d'entendre, i és un tema que s'hauria d'estudiar una miqueta més, ja que les velocitats de throughput són molt importants a la nostra emulació.

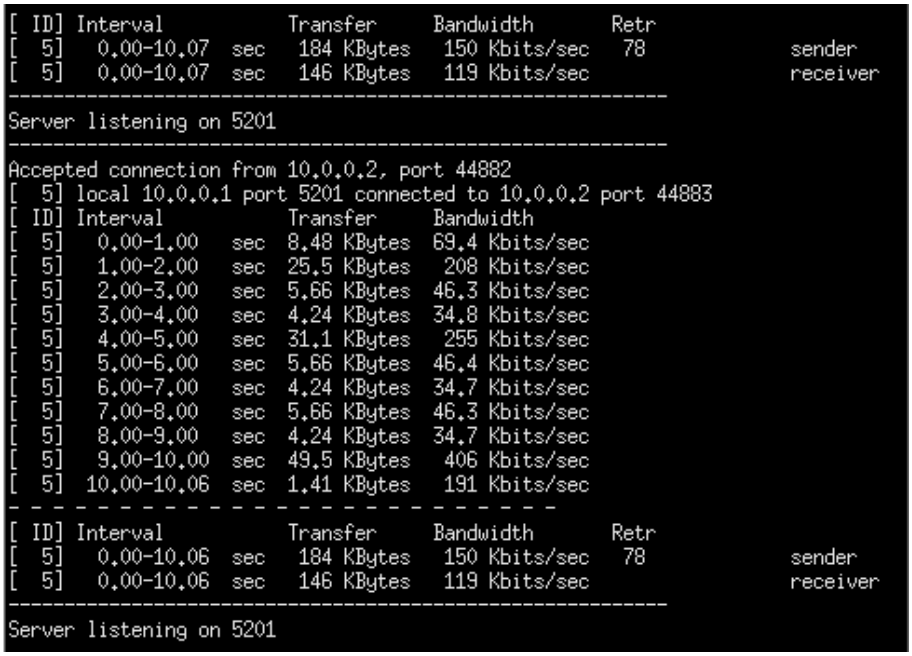



Figura 65: 54Mb/s i 20000 nanosegons de retard = 119-150Kb/s

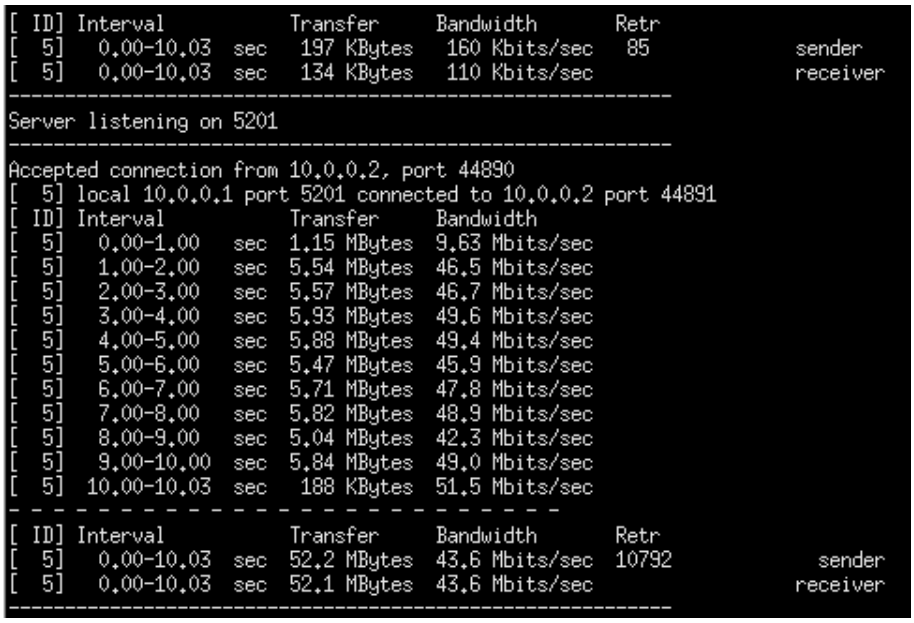



Figura 66: 54Mb/s i 10 nanosegons de retard = 43.6Mb/s

També ens interessaria simular el mateix retard que la xarxa mesh

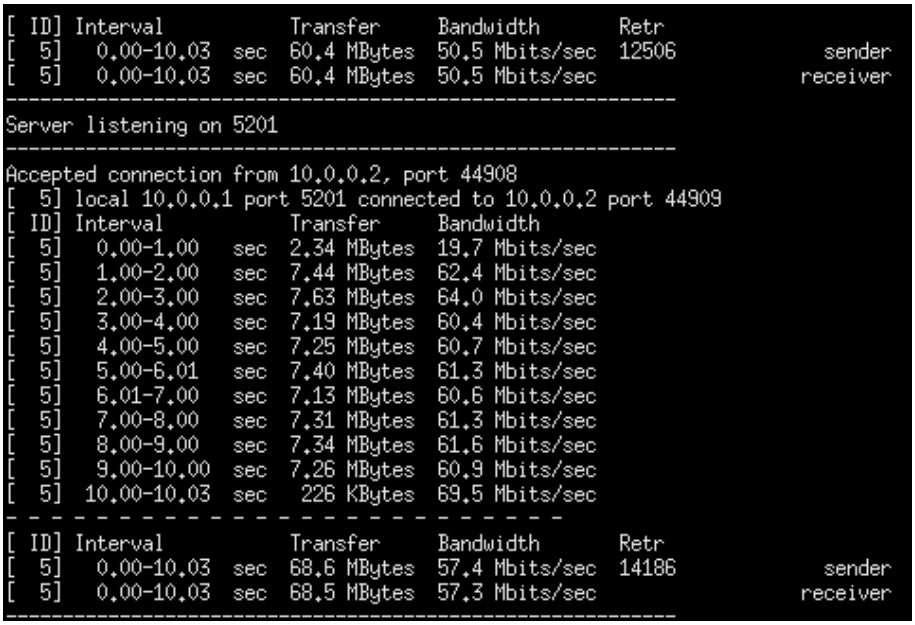



Figura 67: 80Mb/s i 10 nanosegons de retard = 57.4Mb/s

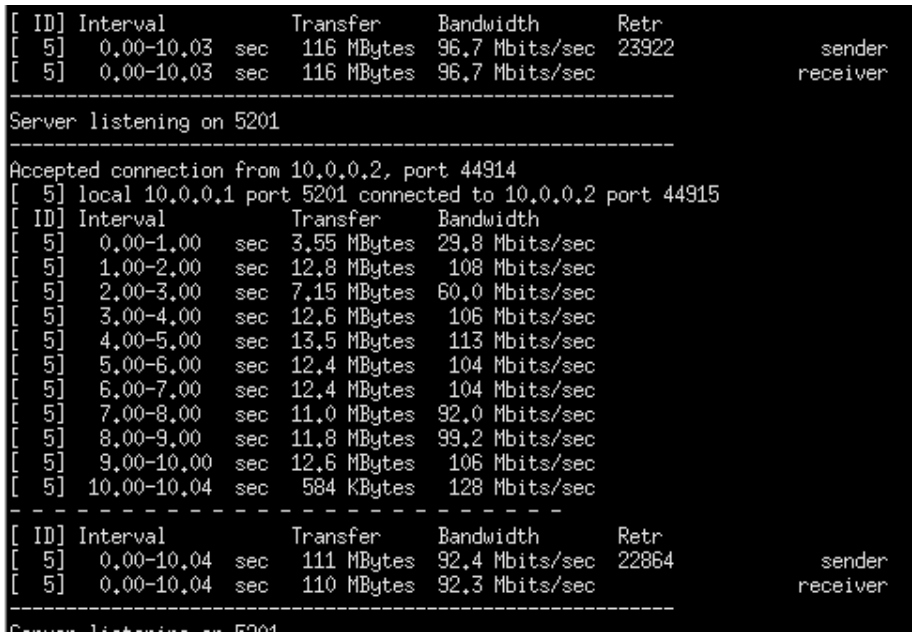



Figura 68: 300Mb/s i 10 nanosegons de retard = 92Mb/s

Guifi.Sants, ja que disposem d'aquesta informació al JSON que fem servir per a convertir la xarxa, aleshores podríem tenir un link exactament igual que a la xarxa real, amb el mateix ample de banda i

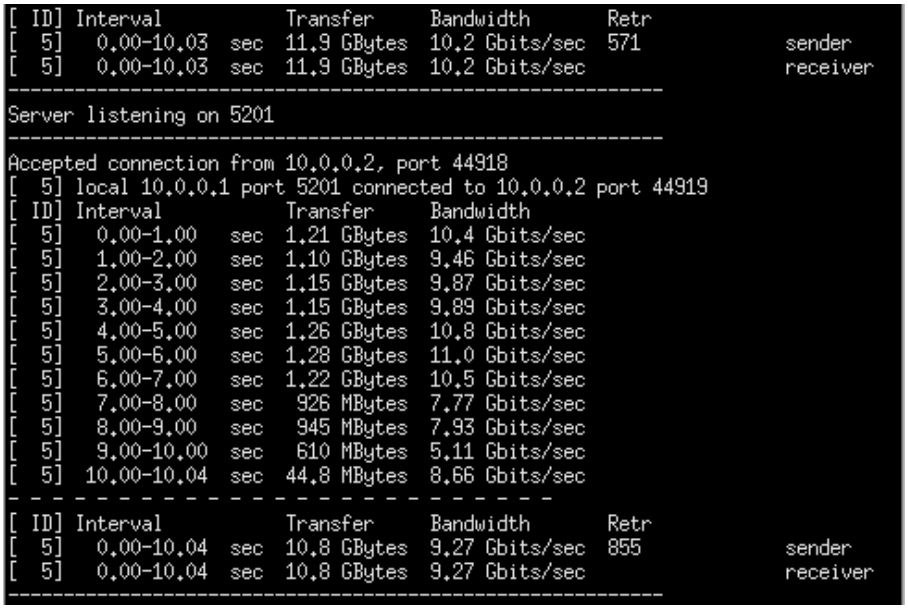



Figura 69: Velocitat il·limitada i 10 nanosegons de retard = 9-10Mb/s

retard. Veiem, però, que si toquem el delay comença a variar bastant la velocitat. Es per això que a la següent simulació, amb els enllaços estàtics, posarem el delay a 10 nanosegons, i deixarem com a future work configurar el mateix retard que a la xarxa de Sants.

## 17.3. Throughput amb enllaços estàtics

A la següent prova, avaluarem la simulació amb enllaços estàtics, configurant, per a cada enllaç, el mateix ample de banda que a la xarxa real, extret del JSON. Agafarem la informació al script i per a cada adjacència entre dos nodes, configurarem el seu bandwith (figura 70). Com a observació, també seria possible agafar el valor RTT"que veiem a la figura 71 (extreta de la pàgina oficial de Guifi.Sants d'on agafem el JSON), i configurar el delay de cada enllaç. Però com ja hem vist que amb retards com 20ms els throughputs no funcionen massa bé, així que provarem els enllaços amb delay de 10 nanosegons.

S'executa la simulació, i es fan proves des de dos nodes adjacents, i també des de 2 nodes el més allunyats possible. S'adjunta un vídeo a la entrega del projecte *throupghput_estatic_ambbw.mp4* en el qual es mostra la prova de throughput.

```
bw = str(int(float(data[2])*1000000))
outfile.write("node n" + str(y+90) + " {\n"
              "    type wlan\n"
              "    network-config {\n"
              " hostname wlan" + str(y) + "\n"
              " !\n"
              " interface wirelesss\n"
              "  ip address 10.0.0.0/8\n"
              " !\n"
              " mobmodel\n"
              " coreapi\n"
              " basic_range\n"
              " !\n"
              "    }\n"
              "    custom-config {\n"
              " custom-config-id basic_range\n"
              " custom-command {3 3 9 9 9}\n"
              " config {\n"
              " range=150000.0\n"
              " bandwidth=" + bw + "\n"
              " jitter=0\n"
```

Figura 70: Línia de codi que afegeix el bandwith de cada enllaç

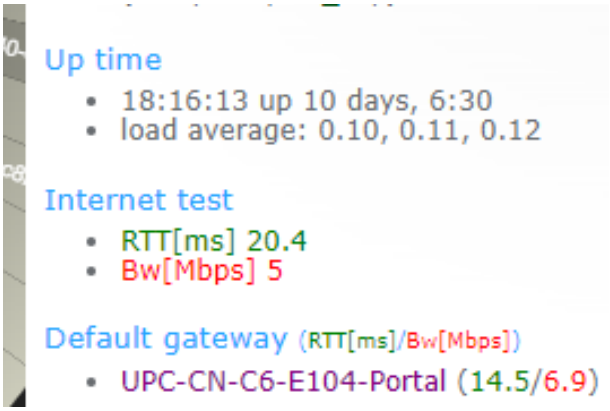



Figura 71: Camp de RTT a cada node

Els resultats no són massa satisfactoris, ja que, encara tenint el retard a 10 nanosegons, i els amples de banda igual que la xarxa real (aquests no baixen de 20Mb/s en el cas més petit), obtenim unes velocitats molt reduïdes. En el cas de nodes adjacents, uns 119-150Kb/s (figura 72), i en nodes allunyats, 38-68Kb/s (figura 73). Es comprova per si de cas, al document de configuració, que els bandwith son correctes. Donats els resultats no seguirem fent proves, ja que podem veure que el throughput no es massa real, i ja havíem trobat molts problemes tant de propagació com de velocitat en aquest escenari amb els enllaços estàtics. Per tant, queda obert per a future work, investigar si en un altre escenari amb més recursos la velocitat s'assimilaria més a la realitat, tal i com hem pogut veure al escenari amb enllaços

dinàmics.

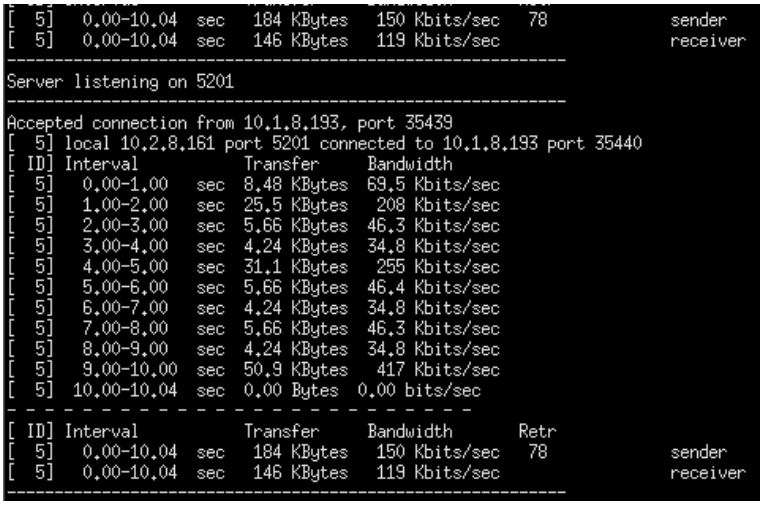



Figura 72: Throughput entre dos nodes adjacents

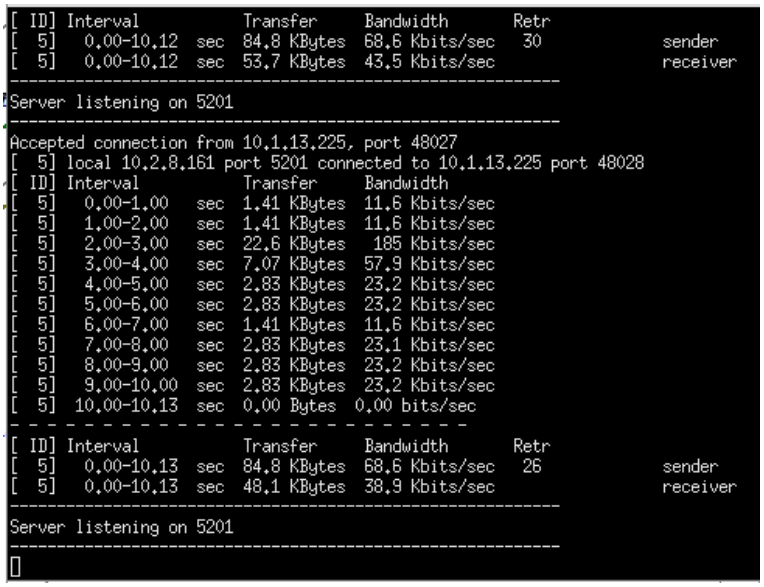



Figura 73: Throughput entre dos nodes adjacents allunyats

# 18. Conclusions

## 18.1. Compliment dels objectius del projecte

Com s'havia esmentat a l'apartat de l'abast, aquests eren els nostres objectius principals del projecte:

- Emular de forma aproximada la topologia de la xarxa mesh de Sants. Representar els enllaços que hi ha entre els nodes, especificant les seves característiques.
- Simular l'algorisme d'encaminament de la xarxa mesh. Idealment BMX6, algorisme que es fa servir a Guifi.Sants, o algun altre de semblant, com ara bé OSPF.

D'aquest dos objectius, podríem dir que el primer s'ha realitzat completament, mentre que el segon, només una part. Donada la dificultat per a poder arribar a córrer qMp, sistema que corre als nodes de Guifi.Sants, el qual conté l'algorisme d'encaminament BMX6, s'ha deixat per a *future work* aquesta implementació, junt amb altres idees que es comentaran en aquest resum de les conclusions.

## 18.2. Coneixements aplicats i adquirits

Per a portar a terme aquest projecte s'han aplicat coneixements varis adquirits durant la carrera, i també de molts s'han adquirit. A continuació mostrem algunes competències tècniques aplicades a aquest projecte, extretes de la pàgina web de la Facultat d'Informàtica de Barcelona:

- **CTI1.2:** *Seleccionar, dissenyar, desplegar, integrar gestionar xarxes i infraestructures de comunicacions en una organització.* Entendre la xarxa mesh sense fils de Guifi.Sants i dissenyar la seva configuració al emulador CORE, ens ha fet entendre en profunditat la seva estructura, i poder arribar a representar-la i entendre el seu funcionament.
- **CTI1.4:** *Seleccionar, dissenyar, desplegar, integrar, avaluar, construir, gestionar, explotar i mantenir les tecnologies de hardware, software i xarxes, dintre dels paràmetres de cost i qualitat adequats.* El fet d'haver investigat amb l'emulador CORE a fons totes les seves característiques, funcionalitats, avantatges i desavantatges, fan que aquest punt sigui un dels que s'ha aplicat més en profunditat.
- **CTI3.1:** *Concebre sistemes, aplicacions i serveis basats en tecnologies de xarxa, tenint en compte Internet, web, comerç electrònic, multimèdia, serveis interactius i computació ubiqua.* Aquest punt

s'ha aplicat en certa mesura, ja que s'han hagut de crear molts scripts d'automatització per a tal de crear els recursos a l'emulador, i després poder executar proves dins d'aquest.

## 18.3. Valoració dels resultats

La valoració general dels resultats obtinguts en general és bastant bona. Hi han molts punts els quals es veuen limitacions al simulador, però també hi han d'altres que es poden explotar bastant.

CORE es un emulador amb bastant de potencial. Hem sigut capaços de representar la xarxa mesh sense fils de Guifi.Sants, representant tots els seus enllaços entre nodes. Hem sigut capaços de representar l'ample de banda de cada enllaç, aspecte molt important per a les simulacions, i també hem definit el retard de l'enllaç, però al final no s'ha acabat implementant al simulador per problemes de rendiment.

Tot i la mida i el número de nodes de la xarxa, s'ha aconseguit córrer moltes simulacions satisfactòriament. Gràcies als contenidors de linux, el consum es mínim per cada node, el qual ens permet aconseguir un millor rendiment global. Hem vist també però que en alguns casos com en el dels enllaços estàtics, la velocitat es veu exponencialment reduïda.

S'ha de tenir en compte que totes les simulacions s'han hagut de córrer al final amb la màquina virtual que ens facilita CORE, ja que intentar instal·lar CORE a altres màquines ha acabat donant problemes. Seguint totes les instruccions de la documentació d'instal·lació, s'instal·la correctament, inclòs els paquets necessaris per els serveis de Quagga i demès. Podem córrer simulacions, però al final sempre falla quelcom, així que també es deixa com a *future work* i com a punt molt interessant d'investigar, poder compilar i tenir CORE al 100 % funcionant a un sistema més preparat per a les simulacions, amb més recursos de hardware.

Un aspecte negatiu de l'emulador es el fet de no poder definir l'abast de la senyal de cada node per separat, i tampoc definir si la senyal es unidireccional o omindireccional. A la xarxa de Guifi.Sants molts nodes son unidireccionals i apunten a una altre antena en concret, però a CORE només podem triar omnidireccional. Donat això la creació d'enllaços automàtica mai serà com a l'entorn real, ja que, apart de no poder definir el tipus de senyal, no es pot definir l'abast individualment per a cada node. Es per això que en aquest aspecte CORE no pot simular l'entorn de la xarxa mesh sense fils Guifi.Sants a l'hora de crear els enllaços automàtics.

El que si que podem emular però, son els enllaços i les seves característiques, tal i com hem comentat anteriorment. Hem sigut capaços de mesurar el throughput general de cada node i també entre dos nodes en un moment determinat, en molts casos amb resultats satisfactoris. Cal investigar però també el tema del retard als enllaços, ja que ens hem vist obligats a posar 10 nanosegons a tots, i això tampoc es realista. El problema al representar l'escenari amb els enllaços estàtics ha sigut el rendiment de la simulació, la qual s'ha vist afectada notablement, produint uns resultats incorrectes.

Donada aquesta lentitud de simulació, seria molt interessant poder córrer CORE a una màquina amb els recursos necessaris per a que la simulació sigui el més realista possible. Aquest concepte s'afegirà com a *future work*.

## 18.4. Future work

Donada la dificultat d'alguns aspectes de l'emulador, i donat també que la documentació no és massa extensa i no hi han molts estudis al respecte, seria interessant estudiar els punts següents en un futur:

- **Implementar qMp als nodes de CORE:** Seria molt interessant poder córrer qMp als nodes de CORE, ja que fariem servir exactament el mateix algorisme d'encaminament que la xarxa de Guifi.Sants (BMX6). També es podria actualitzar a la versió BMX7 dins de qMp i així provar el seu rendiment a una simulació.
- **Compilar i executar CORE a un host més recursos:** Aquesta funcionalitat possiblement seria la que marqués la diferencia, ja que ens permetria tenir els recursos hardware necessaris per a veure si realment el problema a l'escenari estàtic es de rendiment. Si això fos així podríem córrer simulacions més realistes.

## 18.5. Valoració personal del projecte

La meva experiència durant el transcurs d'aquest projecte ha sigut molt satisfactòria. Malgrat haver de treballar amb tecnologies no molt documentades i amb pocs estudis al respecte, al final he obtingut resultats dels quals podem treure conclusions respecte a CORE. Sempre m'han agradat les xarxes i el concepte de xarxa comunitària no el coneixia a fons, i m'ha donat una oportunitat per entendre com funciona.

M'hagués agradat polir alguns aspectes del projecte com ara bé la planificació i fer mes proves, però al final el temps es limitat. En resum estic bastant satisfet amb el resultat.

# 19. Media adjuntat

A l'entrega del projecte s'han adjuntat alguns fitxers i vídeos sobre les simulacions, al fitxer coresants.rar:

- **convert_c.py:** Script en Python per a convertir el JSON de Guifi.sants amb rutes automàtiques en format .imn a CORE
- **convert_wlansbw2_c.py:** Script en Python per a convertir el JSON de Guifi.sants amb rutes estàtiques en format .imn a CORE
- **guifisants.imn:** Fitxer de configuració CORE d'exemple amb la xarxa de Guifi.sants amb enllaços estàtics.
- **sants.json:** Fitxer JSON extret de la pàgina de qMp-Sants
- **autolinks.mp4:** Video de simulació amb links automàtics
- **estatic_links.mp4:** Vídeo de simulació amb links estàtics
- **ospf_autolinks.mp4:** Vídeo de simulació amb links automàtics + convergència OSPF
- **ospf_ping.mp4:** Vídeo de simulació amb links automàtics + convergència OSPF per ping
- **throughput_dinamic:** Video de simulació de throughput amb enllaços automàtics
- **throughput_widget:** Widget de throughput a temps real en nodes
- **throupghput_estatic_ambbw:** Vídeo de simulació de throughput amb enllaços estàtics

## Referencias


[1] «*guifi.net - Xarxa de telecomunicacions de comuns, oberta, lliure i neutral — guifi.net*» Accedit 11 de abril de 2018. https://guifi.net/

[2] *Vega, Davide, Roger Baig, Llorenç Cerdà-Alabern, Esunly Medina, Roc Meseguer, y Leandro Navarro. «A Technological Overview of the Guifi.Net Community Network». Computer Networks 93 (diciembre de 2015): 260-78.* Accedit 11 de abril de 2018. https://doi.org/10.1016/j.comnet.2015.09.023

[3] «*El Comuns de la Xarxa Oberta, Lliure i Neutral ("XOLN") — guifi.net»)* Accedit 11 de abril de 2018. https://guifi.net/ComunsXOLN#ComunsXOLN.

[4] «*Què és guifi·net? — sants.guifi.net*» Accedit 11 de abril de 2018. http://sants.guifi.net/guifinet

[5] «*Open Shortest Path First*». Accedit 11 de abril de 2018. https://www.cisco.com/c/en/us/products/ios-nx-os-software/open-shortest-path-first-ospf/index.html

[6] *Tremback, Jehan. «Using C.O.R.E. to work on mesh network routing protocols». Althea.* Accedit 11 de abril de 2018. http://altheamesh.com/blog/using-core-for-network-simulation/

[7] «*Resum - bmx6 - qMp - Quick Mesh Project / BMX6 - BMX7 development site*». Accedit 11 de abril de 2018. http://bmx6.net/projects/bmx6.

[8] «*Xarxa Sense Fils Cooperativa*». Accedit 11 de abril de 2018. https://xsf-coop.net/PAGINDEX-CAT.html

[9] «*What Is Freifunk about? > Freifunk.Net*». Accedit 11 de abril de 2018. https://freifunk.net/en/what-is-it-about/

[10] «*¿Quiénes somos? — Grupo de usuarios de Software Libre Rosario*». Accedit 11 de abril de 2018. http://www.lugro.org.ar/quienes_somos

[11] *Neumann, Axel, Ester Lopez, y Leandro Navarro. «An Evaluation of BMX6 for Community Wireless Networks», 651-58. IEEE, 2012* Accedit 11 de abril de 2018. https://doi.org/10.1109/WiMOB.2012.6379145

[12] «*Documentation - qMp*». Accedit 11 de abril de 2018. http://qmp.cat/Documentation

[13] «*OpenWrt Project: Welcome to the OpenWrt Project*». Accedit 11 de abril de 2018. https://openwrt.org/start

[14] «*U.S. Naval Research Laboratory (NRL)*». Accedit 11 de abril de 2018. https://www.nrl.navy.mil/

[15] «*qMp Sants-UPC — Pàgina de monitorització de la xarxa Guifi.sants*». Accedit 19 de juny de 2018. http://tomir.ac.upc.edu/qmpsu/index.php

# 20. Annexos

## 20.1. Codi per a generar xarxa amb enllaços dinàmics

```
import urllib,json
from pprint import pprint
import unicodedata
import socket
import struct

url = "http://tomir.ac.upc.edu/qmpsu/download_json.php"
response = urllib.urlopen(url)
data = json.loads(response.read())

outfile = open('guifisants.imn','w')

networks = []
dictnetworks = dict()
listnodes = []

i = 0

#nom de node de la wlan
print str(len(data['nodeList']))

for node in data['nodeList']:

    name = unicodedata.normalize('NFKD', node['name']).encode('ascii','ig

    #dades del node
    id = unicodedata.normalize('NFKD', node['id']).encode('ascii','ignore
    nodename = "n" + str(i+1)
    listnodes.append(nodename)

    outfile.write("node " + nodename + " {\n"
    "    type router\n"
    "    model router\n"
    "    network-config {\n"
    "    hostname " + name + "\n"
    "    !\n")

    if 'ipv4' in node['data']:
        #recorrer interficies per trobar la IP /32 que es la que volem
        for interface in node['data']['ipv4']:
```

```
            v4 = unicodedata.normalize('NFKD', interface).encode('ascii',

            if (v4.split('/')[1] == "32"):
                ipv4 = v4

    #assignem la ip /32
    outfile.write(
    "    interface eth0\n"
    "     ip address " + ipv4 + "\n"
    "    !\n")

    #coordenades /10
    if 'x' in node:
        cordx = node['x']-9000
        cordy = node['y']-9000
    if cordx < 0:
        cordx = 0
    if cordy < 0:
        cordy = 0

    outfile.write("    }\n"
                  "    canvas c1\n"
                  "    iconcoords {" + str(format(cordx,'.1f')) + " " + s
                  "    labelcoords {" + str(format(cordx,'.1f')) + " " +
                  "    interface-peer {eth0 n999}\n"
                  "    custom-config {\n"
                  "    custom-config-id service:zebra:/usr/local/etc/qua
                  "    custom-command /usr/local/etc/quagga/Quagga.conf\
                  "    config {\n"
                  "    interface eth0\n"
                  "     ip address " + ipv4 + "\n"
                  "    !\n"
                  "    router ospf6\n"
                  "     router-id " + ipv4.split('/')[0] + "\n"
                  "     interface eth0 area 0.0.0.0\n"
                  "     redistribute connected\n"
                  "     redistribute ospf\n"
                  "    !\n"
                  "    }\n"
                  "    }\n"
                  "    custom-config {\n"
                  "    custom-config-id service:UserDefined\n"
                  "    custom-command UserDefined\n"
```

```
                "␣␣␣␣␣config␣{\n"
                "␣␣␣␣␣files=('mgen.sh',␣)\n"
                "␣␣␣␣␣startidx=35\n"
                "␣␣␣␣␣cmdup=('sh␣mgen.sh',␣)\n"
                "␣␣␣␣␣}\n"
                "␣␣␣␣}\n"
                "␣␣␣␣custom-config␣{\n"
                "␣␣␣␣␣custom-config-id␣service:UserDefined:mgen.sh\n"
                "␣␣␣␣␣custom-command␣mgen.sh\n"
                "␣␣␣␣␣config␣{\n"
                "␣␣␣␣␣#!/bin/sh\n"
                '␣␣uptime␣|␣cut␣-d"␣"-f2␣>>␣outfile.txt\n'
                '␣␣result="$(ifconfig␣|␣grep␣10.1.15.129)"\n'
                '␣␣if␣[␣"$result"␣!=␣""␣];␣then\n'
                '␣␣␣nc␣-k␣-l␣2399␣>>␣outfile.txt␣&␣\n'
                '␣␣fi\n'
                '␣␣routes="$(ip␣r␣|␣grep␣zebra␣|␣wc␣-l)"\n'
                "␣␣while␣[␣$routes␣-lt␣75␣]\n"
                "␣␣do\n"
                "␣␣␣sleep␣10\n"
                '␣␣␣routes="$(ip␣r␣|␣grep␣zebra␣|␣wc␣-l)"\n'
                "␣␣done\n"
                '␣␣uptime␣|␣cut␣-d"␣"-f2␣|␣nc␣10.1.15.129␣2399\n'
                "␣␣␣␣␣}\n"
                "␣␣␣␣}\n"
                "␣␣␣␣services␣{zebra␣OSPFv2␣OSPFv3MDR␣vtysh␣UserDefined
                "}\n\n")


    i+=1

outfile.write("node␣n999␣{\n"
              "␣␣␣␣type␣wlan\n"
              "␣␣␣␣network-config␣{\n"
              "␣hostname␣wlan999\n"
              "␣!\n"
              "␣interface␣wirelesss\n"
              "␣␣ip␣address␣10.0.0.0/8\n"
              "␣!\n"
              "␣mobmodel\n"
              "␣coreapi\n"
              "␣basic_range\n"
              "␣!\n"
              "␣␣␣␣}\n"
```

```
                    "␣␣␣␣custom-config␣{\n"
                    "␣custom-config-id␣basic_range\n"
                    "␣custom-command␣{3␣3␣9␣9␣9}\n"
                    "␣config␣{\n"
                    "␣range=1500.0\n"
                    "␣bandwidth=54000000\n"
                    "␣jitter=0\n"
                    "␣delay=10\n"
                    "␣error=0\n"
                    "␣}\n"
                    "␣␣␣␣}\n"
                    "␣␣␣␣canvas␣c1\n"
                    "␣␣␣␣iconcoords␣{200.0␣200.0}\n"
                    "␣␣␣␣labelcoords␣{230.0␣236.0}\n")

y = 0
for peer in listnodes:
    outfile.write("␣␣␣␣interface-peer␣{e" + str(y) + "␣" + peer + "}\n")
    y += 1

outfile.write("}\n\n")

y = 1
for peer in listnodes:
    outfile.write("link␣l" + str(y) + "␣{\n"
                  "␣␣␣␣nodes␣{n999␣"  + peer + "}\n"
                  "}\n\n")
    y += 1


outfile.write("canvas␣c1␣{\n"
"␣␣␣␣name␣{Canvas1}\n"
"}\n\n"

"option␣global␣{\n"
"␣␣␣␣interface_names␣no\n"
"␣␣␣␣ip_addresses␣yes\n"
"␣␣␣␣ipv6_addresses␣yes\n"
"␣␣␣␣node_labels␣yes\n"
"␣␣␣␣link_labels␣yes\n"
"␣␣␣␣show_api␣no\n"
"␣␣␣␣background_images␣no\n"
"␣␣␣␣annotations␣yes\n"
"␣␣␣␣grid␣yes\n"
```

```
"    traffic_start 0\n"
"}\n\n"

"option session {\n"
"}\n")


outfile.close()
```

### 20.2. Codi per a generar xarxa amb enllaços estàtics

```
import urllib,json
from pprint import pprint
import unicodedata
import socket
import struct

url = "http://tomir.ac.upc.edu/qmpsu/download_json.php"
response = urllib.urlopen(url)
data = json.loads(response.read())

outfile = open('guifisants.imn','w')


adjacencies = []
adjacenciesbw= []
maxx = maxy = 0
minx = miny = 15000
xx = 0


i = 0

#nom de node de la wlan
wlan = "n" + str(len(data['nodeList']))

for node in data['nodeList']:

    name = unicodedata.normalize('NFKD', node['name']).encode('ascii','ig
    id = unicodedata.normalize('NFKD', node['id']).encode('ascii','ignore

    #descartem el node de castelldefels


    id = int(id.split('00')[1])
    id = "n" + str(id)


    outfile.write("node " + id + " {\n"
    "    type router\n"
    "    model router\n"
    "    network-config {\n"
    "    hostname " + name + "\n"
```

```
"␣␣␣!\n")

ipv4 = "0.0.0.0"

if 'ipv4' in node['data']:
    #recorrer interficies per trobar la IP /32 que es la que volem
    for interface in node['data']['ipv4']:

        v4 = unicodedata.normalize('NFKD', interface).encode('ascii',

        if (v4.split('/')[1] == "32"):
            ipv4 = v4

if ('adjacencies' in node):
    interfaces = len(node['adjacencies'])
    j = 0

    #assignem la ip /32 per a tantes adjacencies com tingui el node
    while (j < interfaces):
        outfile.write(
        "␣␣␣interface␣eth" + str(j) + "\n"
        "␣␣␣␣ip␣address␣" + ipv4 + "\n"
        "␣␣␣!\n")
        j += 1
else:
    outfile.write(
    "␣␣␣␣␣␣␣interface␣eth0\n"
    "␣␣␣␣␣␣␣␣ip␣address␣" + ipv4 + "\n"
    "␣␣␣␣␣␣␣!\n")

#coordenades /10
if 'x' in node:
    cordx = node['x']-9000
    cordy = node['y']-9000

    if (cordx > maxx):
        maxx = cordx
    if (cordy > maxy):
        maxy = cordy

    if (cordx < minx):
        minx = cordx
    if (cordy < miny):
        miny = cordy
```

```
outfile.write("    }\n"
              "    canvas c1\n"
              "    iconcoords {" + str(format(cordx,'.1f')) + " " + s
              "    labelcoords {" + str(format(cordx,'.1f')) + " " +


#adjacencies del node
if ('adjacencies' in node):
    z = 0
    for adj in node['adjacencies']:

        nodeTo = unicodedata.normalize('NFKD', adj['nodeTo']).encode(
        if ('bw' in adj['data']):
            bw = adj['data']['bw']
        nodeTo = int(nodeTo.split('00')[1])
        current = id + "/" + "n" + str(nodeTo)
        current_back = "n" + str(nodeTo) + "/" + id

        #si ja sha guardat el link anteriorment no el guardarem a la
        if current_back not in adjacencies:
            adjacencies.append(current)
            adjacenciesbw.append(current + "/" + str(bw))
            outfile.write("    interface-peer {eth" + str(z) + " n" +
        else:
            outfile.write("    interface-peer {eth" + str(z) + " n" +
        z += 1

outfile.write(
              "    custom-config {\n"
              "    custom-config-id service:zebra:/usr/local/etc/qua
              "    custom-command /usr/local/etc/quagga/Quagga.conf\
              "    config {\n"
              "    interface eth0\n"
              "        ip address " + ipv4 + "\n"
              "    !\n"
              "    router ospf6\n"
              "        router-id " + ipv4.split('/')[0] + "\n"
              "        interface eth0 area 0.0.0.0\n"
              "        redistribute connected\n"
              "        redistribute ospf\n"
              "    !\n"
              "    }\n"
              "    }\n"
```

```
                    "    custom-config {\n"
                    "      custom-config-id service:UserDefined\n"
                    "      custom-command UserDefined\n"
                    "      config {\n"
                    "        files=('mgen.sh', )\n"
                    "        startidx=35\n"
                    "        cmdup=('sh mgen.sh', )\n"
                    "      }\n"
                    "    }\n"
                    "    custom-config {\n"
                    "      custom-config-id service:UserDefined:mgen.sh\n"
                    "      custom-command mgen.sh\n"
                    "      config {\n"
                    "        #!/bin/sh\n"
                    '  uptime | cut -d" " -f2 >> outfile.txt\n'
                    '  result="$(ifconfig | grep 10.1.15.129)"\n'
                    '  if [ "$result" != "" ]; then\n'
                    '    nc -k -l 2399 >> outfile.txt &\n'
                    '  fi\n'
                    '  routes="$(ip r | grep zebra | wc -l)"\n'
                    "  while [ $routes -lt 75 ]\n"
                    "  do\n"
                    "    sleep 10\n"
                    '    routes="$(ip r | grep zebra | wc -l)"\n'
                    "  done\n"
                    '  uptime | cut -d" " -f2 | nc 10.1.15.129 2399\n'
                    "      }\n"
                    "    }\n"
                    "    services {zebra OSPFv2 OSPFv3MDR vtysh UserDefined
                    "}\n\n")


    i+=1

#creem tantes wlans com adjacencies entre nodes
y = 0
for wlans in adjacenciesbw:
    data = wlans.split('/')
    peer1 = data[0]
    peer2 = data[1]
    bw = str(int(float(data[2])*1000000))
    outfile.write("node n" + str(y+90) + " {\n"
                  "    type wlan\n"
                  "    network-config {\n"
```

```
                    "     hostname wlan" + str(y) + "\n"
                    "     !\n"
                    "     interface wirelesss\n"
                    "      ip address 10.0.0.0/8\n"
                    "     !\n"
                    "     mobmodel\n"
                    "     coreapi\n"
                    "     basic_range\n"
                    "     !\n"
                    "    }\n"
                    "    custom-config {\n"
                    "     custom-config-id basic_range\n"
                    "     custom-command {3 3 9 9 9}\n"
                    "     config {\n"
                    "     range=150000.0\n"
                    "     bandwidth=" + bw + "\n"
                    "     jitter=0\n"
                    "     delay=20000\n"
                    "     error=0\n"
                    "     }\n"
                    "    }\n"
                    "    canvas c1\n"
                    "    iconcoords {" + str(100+xx) + " 200.0}\n"
                    "    labelcoords {" + str(130+xx) + " 236.0}\n"
                    "    interface-peer {eth0 " + peer1 + "}\n"
                    "    interface-peer {eth1 " + peer2 + "}\n"
                    "}\n\n")
    y += 1
    xx += 25

#creem tants links com adjacencies entre nodes
y = 1
z = 90
for peer in adjacencies:
    peer1 = peer.split('/')[0]
    peer2 = peer.split('/')[1]
    outfile.write("link l" + str(y) + " {\n"
                  "    nodes {n" + str(z) + " " + peer1 + "}\n"
                  "}\n\n"
                  "link l" + str(y+1) + " {\n"
                  "    nodes {n" + str(z) + " " + peer2 + "}\n"
                  "}\n\n")
    y += 2
    z += 1
```

```
outfile.write("canvas c1 {\n"
"    name {Canvas1}\n"
"}\n\n"

"option global {\n"
"    interface_names no\n"
"    ip_addresses yes\n"
"    ipv6_addresses yes\n"
"    node_labels yes\n"
"    link_labels yes\n"
"    show_api no\n"
"    background_images no\n"
"    annotations yes\n"
"    grid yes\n"
"    traffic_start 0\n"
"}\n\n"

"option session {\n"
"}\n")


outfile.close()
```

## 20.3. Pàgina de Guifi.Sants qMp

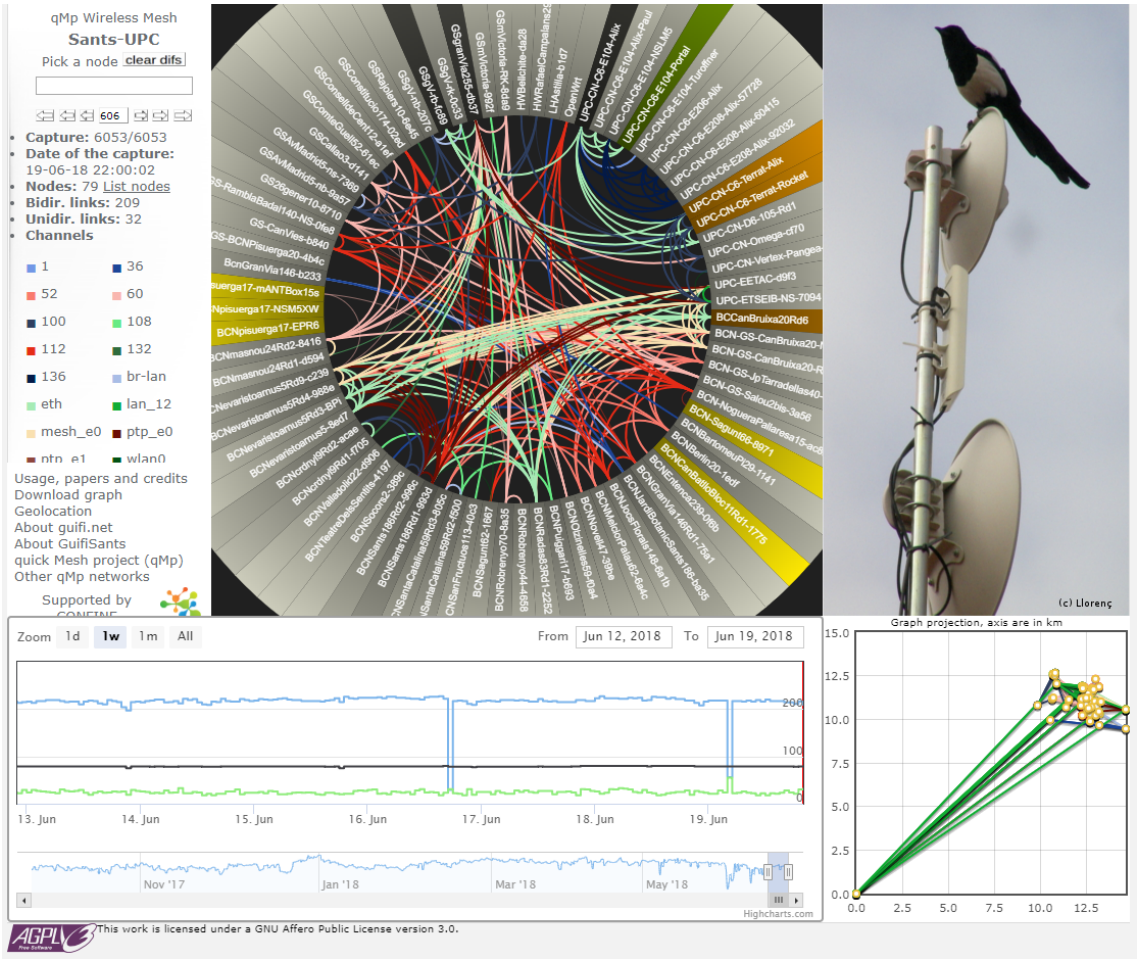



Figura 74: La pàgina conté informació en temps real sobre tots els nodes i enllaços de la xarxa Guifi.sants